Sponsored by
National Science Foundation
Advanced Technological Education
Award No. DUE-1003389

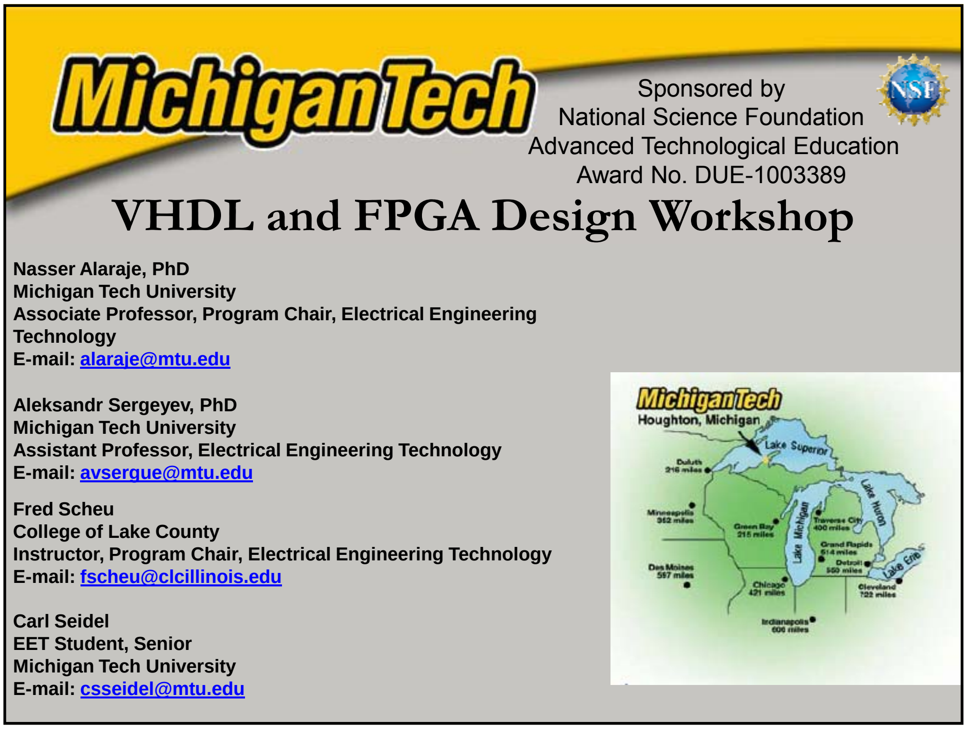
VHDL and FPGA Design Workshop

Nasser Alaraje, PhD
Michigan Tech University
Associate Professor, Program Chair, Electrical Engineering Technology
E-mail: alaraje@mtu.edu

Aleksandr Sergeyev, PhD
Michigan Tech University
Assistant Professor, Electrical Engineering Technology
E-mail: avsergue@mtu.edu

Fred Scheu
College of Lake County
Instructor, Program Chair, Electrical Engineering Technology
E-mail: fscheu@clcillinois.edu

Carl Seidel
EET Student, Senior
Michigan Tech University
E-mail: csseidel@mtu.edu



MichiganTech

Sponsored by
National Science Foundation
Advanced Technological Education
Award No. DUE-1003389

VHDL and FPGA Design Workshop

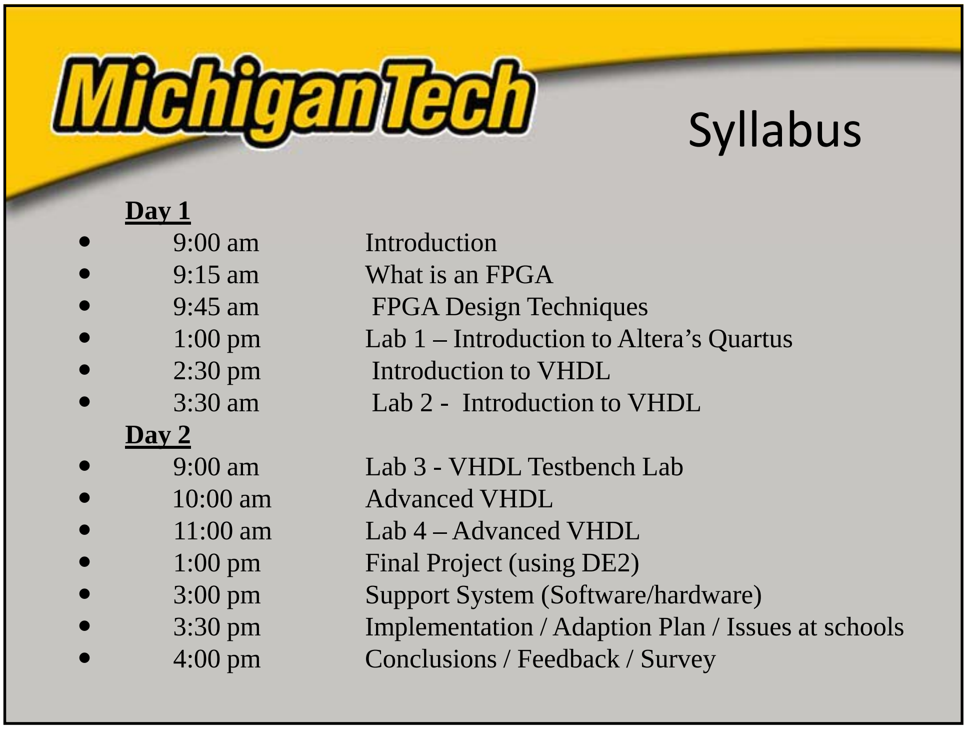
Nasser Alaraje, PhD
Michigan Tech University
Associate Professor, Program Chair, Electrical Engineering Technology
E-mail: alaraje@mtu.edu

Aleksandr Sergeyev, PhD
Michigan Tech University
Assistant Professor, Electrical Engineering Technology
E-mail: avsergue@mtu.edu

Fred Scheu
College of Lake County
Instructor, Program Chair, Electrical Engineering Technology
E-mail: fscheu@clcillinois.edu

Carl Seidel
EET Student, Senior
Michigan Tech University
E-mail: csseidel@mtu.edu

MichiganTech
Houghton, Michigan
Lake Superior
Duluth 216 miles
Minneapolis 352 miles
Green Bay 215 miles
Lake Michigan
Traverse City 400 miles
Lake Huron
Grand Rapids 514 miles
Detroit 550 miles
Lake Erie
Des Moines 597 miles
Chicago 421 miles
Cleveland 722 miles
Indianapolis 606 miles



MichiganTech

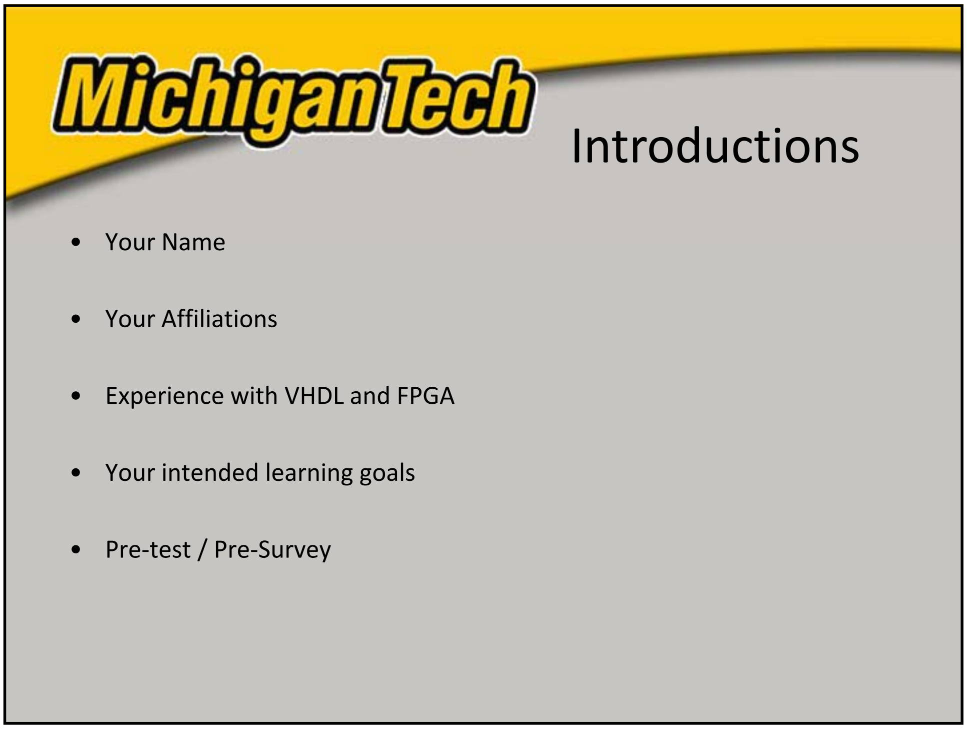
Syllabus

Day 1

9:00 am	Introduction
9:15 am	What is an FPGA
9:45 am	FPGA Design Techniques
1:00 pm	Lab 1 – Introduction to Altera's Quartus
2:30 pm	Introduction to VHDL
3:30 am	Lab 2 - Introduction to VHDL

Day 2

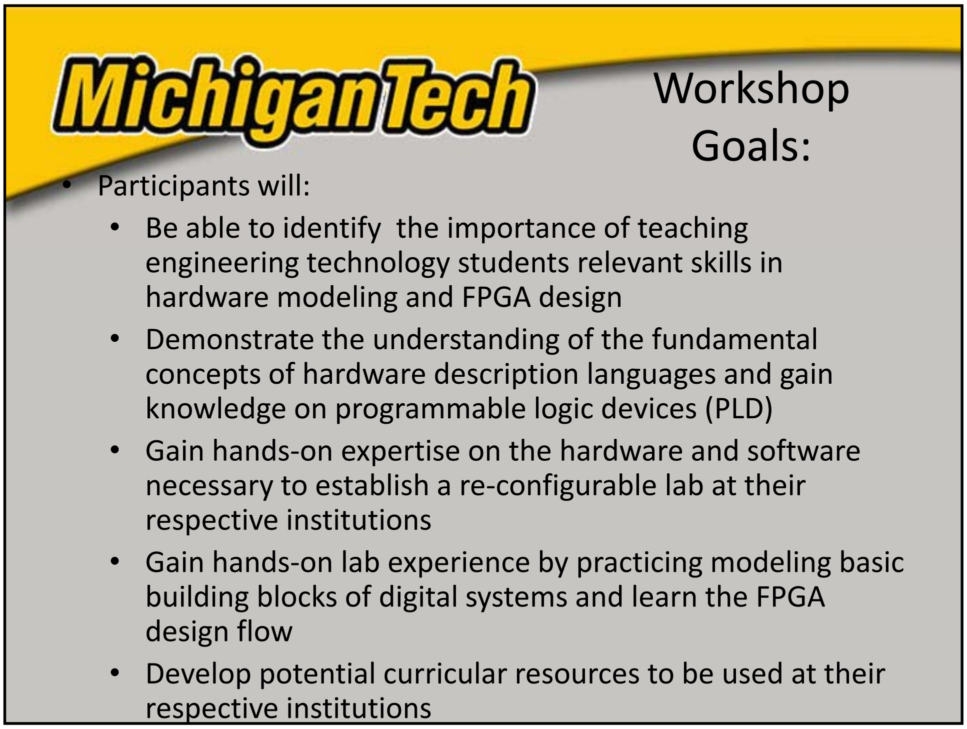
9:00 am	Lab 3 - VHDL Testbench Lab
10:00 am	Advanced VHDL
11:00 am	Lab 4 – Advanced VHDL
1:00 pm	Final Project (using DE2)
3:00 pm	Support System (Software/hardware)
3:30 pm	Implementation / Adaption Plan / Issues at schools
4:00 pm	Conclusions / Feedback / Survey



MichiganTech

Introductions

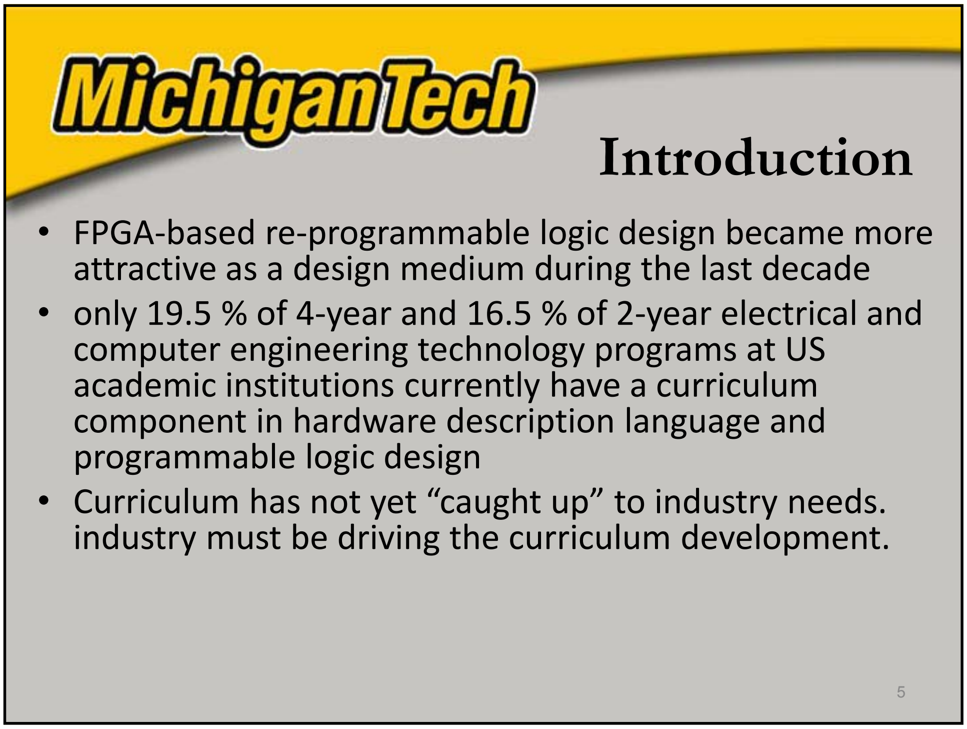
- Your Name
- Your Affiliations
- Experience with VHDL and FPGA
- Your intended learning goals
- Pre-test / Pre-Survey



MichiganTech

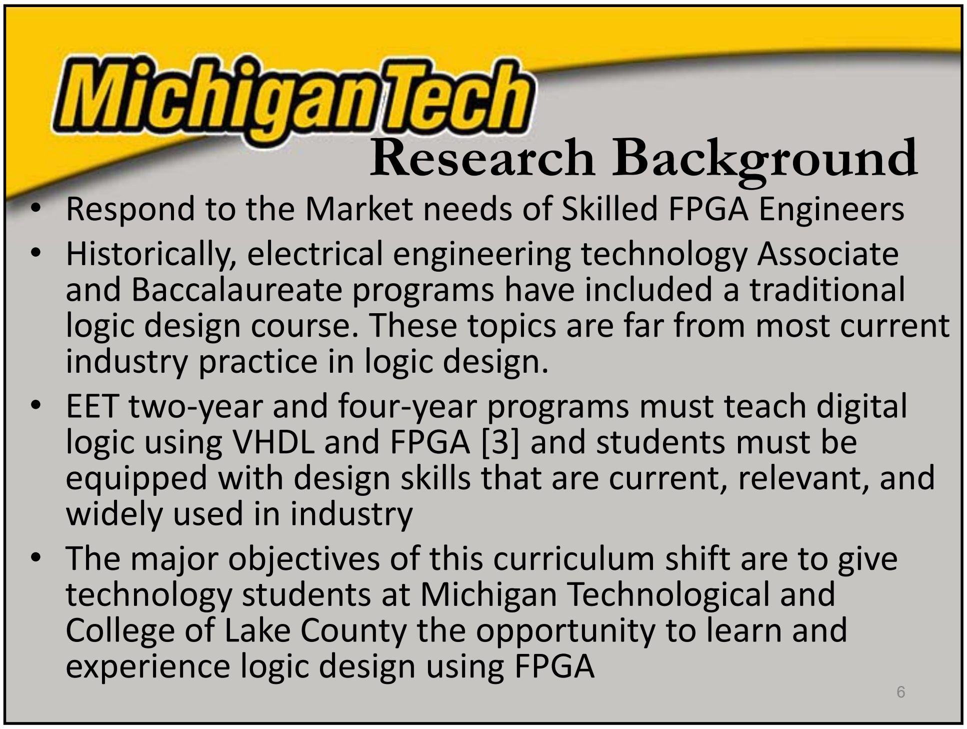
Workshop Goals:

- Participants will:
  - Be able to identify the importance of teaching engineering technology students relevant skills in hardware modeling and FPGA design
  - Demonstrate the understanding of the fundamental concepts of hardware description languages and gain knowledge on programmable logic devices (PLD)
  - Gain hands-on expertise on the hardware and software necessary to establish a re-configurable lab at their respective institutions
  - Gain hands-on lab experience by practicing modeling basic building blocks of digital systems and learn the FPGA design flow
  - Develop potential curricular resources to be used at their respective institutions



Introduction

- FPGA-based re-programmable logic design became more attractive as a design medium during the last decade
- only 19.5 % of 4-year and 16.5 % of 2-year electrical and computer engineering technology programs at US academic institutions currently have a curriculum component in hardware description language and programmable logic design
- Curriculum has not yet “caught up” to industry needs. industry must be driving the curriculum development.

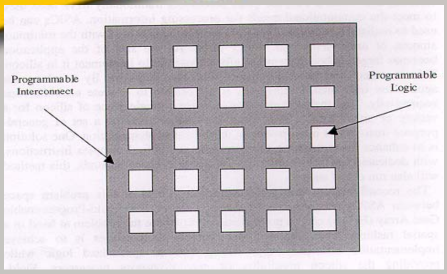


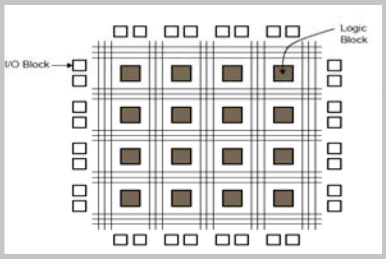
Research Background

- Respond to the Market needs of Skilled FPGA Engineers
- Historically, electrical engineering technology Associate and Baccalaureate programs have included a traditional logic design course. These topics are far from most current industry practice in logic design.
- EET two-year and four-year programs must teach digital logic using VHDL and FPGA [3] and students must be equipped with design skills that are current, relevant, and widely used in industry
- The major objectives of this curriculum shift are to give technology students at Michigan Technological and College of Lake County the opportunity to learn and experience logic design using FPGA

MichiganTech

FPGA





- **Introduced by Xilinx in mid 1980 for implementing digital logic**
  F ield
  P rogrammable
  G ate
  A rray
- **FPGA Can be visualized as a set of programmable logic blocks embedded in programmable interconnect**
- **Interconnect architecture provides the connectivity between logic blocks**
- **Programming Technology determines the method of storing configuration**

MichiganTech

FPGA Re-programmable Logic Applications

- When FPGA first introduced, it was considered as another form of gate array
- SRAM-FPGA in-circuit reprogrammability feature provides a more than just a standard gate array
- FPGAs have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications
  – random logic
  – Custom computing machine
  – device controllers
  – communication encoding and filtering
- programmable logic becomes the dominant form of digital logic design and implementation

MichiganTech

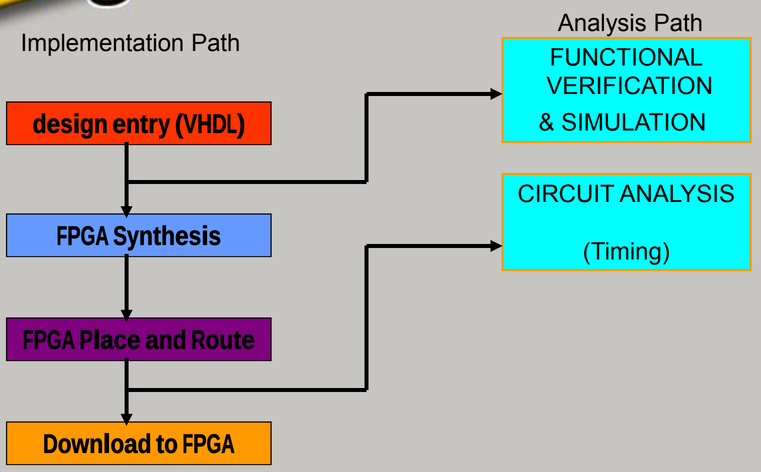
FPGA design flow

- Design Flow is the step-by-step methodology to go through the process of FPGA design
- The design flow can be divided into 6 basic steps

  Design Entry
  Functional Verification and Simulation
  FPGA Synthesis
  FPGA Place & Route
  Circuit Analysis (Timing, Power …)
  Programming FPGA devices

MichiganTech

FPGA Design Flow



MichiganTech

Description of Design steps

- Design Entry – describes the design that has to be implemented onto FPGA
- Functional Verification and Simulation – checks logical correctness of design
- FPGA synthesis – converts design entry into actual gates/blocks needed
- FPGA Place & Route – selects the optimal position and minimizes length of interconnections on device
- Time Analysis – determines the speed of the circuit which has been completely placed and routed
- Programming to FPGA – downloads bitstream codes onto FPGA devices

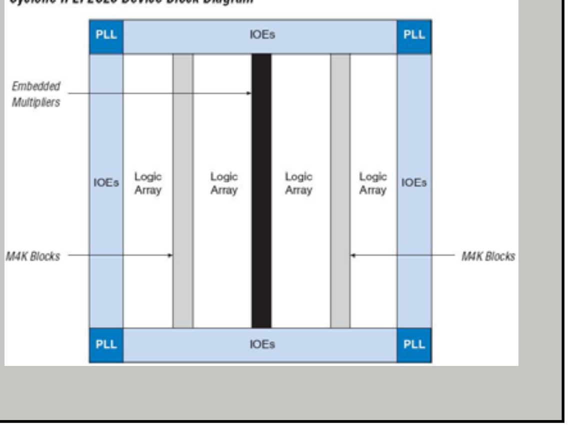
MichiganTech

Cyclone II FPGA Architecture

The Cyclone II FPGA has up to:

- 68,416 logic elements in the logic arrays.
- 150 18x18-bit embedded multipliers.
- 250 M4K RAM blocks
- 622 I/O element pins
- Four phased-locked loops

Cyclone II EP2C20 Device Block Diagram

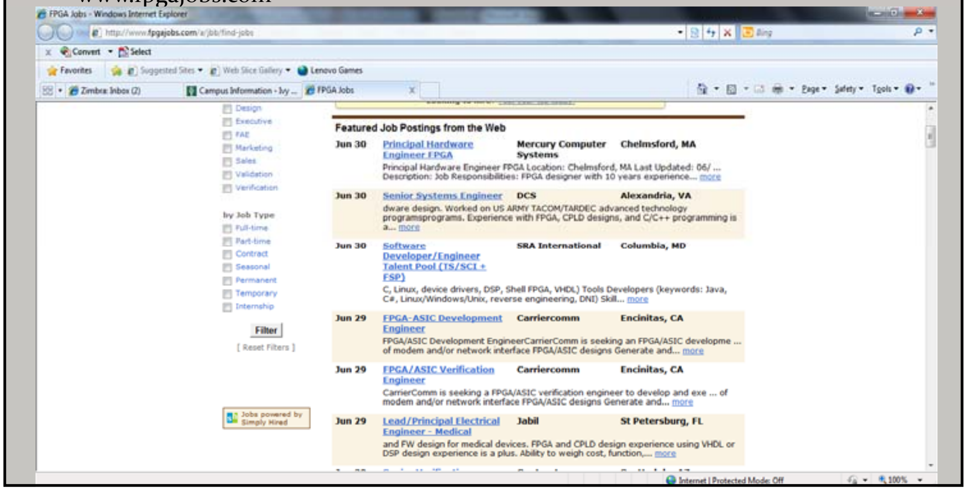


What projects are FPGAs good for

- **Aerospace & Defense**
  Radiation-tolerant FPGAs along with intellectual property for image processing, waveform generation, and partial reconfiguration for SDRs.
- **Automotive**
  Automotive silicon and IP solutions for gateway and driver assistance systems, comfort, convenience, and in-vehicle infotainment.
- **Broadcast**
  Solutions enabling a vast array of broadcast chain tasks as video and audio finds its way from the studio to production and transmission and then to the consumer.
- **Consumer**
  Cost-effective solutions enabling next generation, full-featured consumer applications, such as converged handsets, digital flat panel displays, information appliances, home networking, and residential set top boxes.
- **Industrial/Scientific/Medical**
  Industry-compliant solutions addressing market-specific needs and challenges in industrial automation, motor control, and high-end medical imaging.
- **Storage & Server**
  Data processing solutions for Network Attached Storage (NAS), Storage Area Network (SAN), servers, storage appliances, and more.
- **Wireless Communications**
  RF, base band, connectivity, transport and networking solutions for wireless equipment, addressing standards such as WCDMA, HSDPA, WiMAX and others.
- **Wired Communications**
  End-to-end solutions for the Reprogrammable Networking Linecard Packet Processing, Framer/MAC, serial backplanes, and more

Who uses them

www.fpgajobs.com


FPGA Jobs - Windows Internet Explorer
http://www.fpgajobs.com/a/jbb/find-jobs
Bing
Convert
Select
Favorites
Suggested Sites
Web Slice Gallery
Lenovo Games
Zimbra: Inbox (2)
Campus Information - Ivy ...
FPGA Jobs
Page
Safety
Tools
Internet | Protected Mode: Off
100%

□ Design
□ Executive
□ FAE
□ Marketing
□ Sales
□ Validation
□ Verification

by Job Type

□ Full-time
□ Part-time
□ Contract
□ Seasonal
□ Permanent
□ Temporary
□ Internship

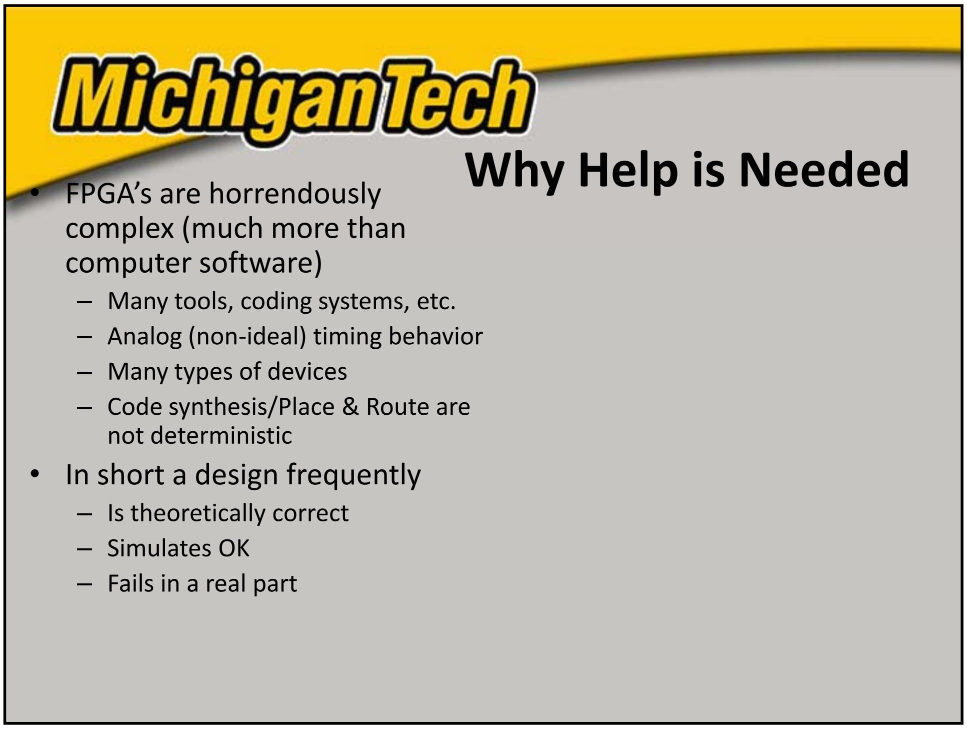
Filter

[Reset Filters]

Jobs powered by Simply Hired

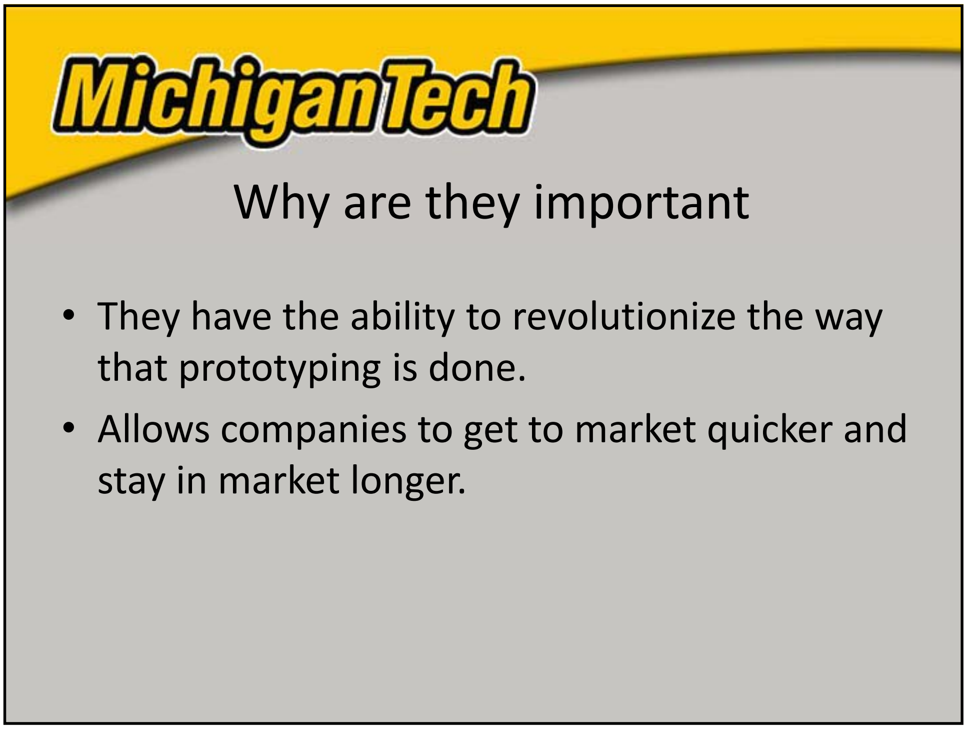
Featured Job Postings from the Web

Jun 30	Principal Hardware Engineer FPGA	Mercury Computer Systems	Chelmsford, MA
	Principal Hardware Engineer FPGA Location: Chelmsford, MA Last Updated: 06/ ... Description: Job Responsibilities: FPGA designer with 10 years experience... more		
Jun 30	Senior Systems Engineer	DCS	Alexandria, VA
	dware design. Worked on US ARMY TACOM/TARDEC advanced technology programsprograms. Experience with FPGA, CPLD designs, and C/C++ programming is a... more		
Jun 30	Software Developer/Engineer Talent Pool (TS/SCI + FSP)	SRA International	Columbia, MD
	C, Linux, device drivers, DSP, Shell FPGA, VHDL) Tools Developers (keywords: Java, C#, Linux/Windows/Unix, reverse engineering, DNI) Skill... more		
Jun 29	FPGA-ASIC Development Engineer	Carriercomm	Encinitas, CA
	FPGA/ASIC Development EngineerCarrierComm is seeking an FPGA/ASIC developme ... of modem and/or network interface FPGA/ASIC designs Generate and... more		
Jun 29	FPGA/ASIC Verification Engineer	Carriercomm	Encinitas, CA
	CarrierComm is seeking a FPGA/ASIC verification engineer to develop and exe ... of modem and/or network interface FPGA/ASIC designs Generate and... more		
Jun 29	Lead/Principal Electrical Engineer - Medical	Jabil	St Petersburg, FL
	and FW design for medical devices. FPGA and CPLD design experience using VHDL or DSP design experience is a plus. Ability to weigh cost, function,... more		



Why Help is Needed

- FPGA's are horrendously complex (much more than computer software)
  - Many tools, coding systems, etc.
  - Analog (non-ideal) timing behavior
  - Many types of devices
  - Code synthesis/Place & Route are not deterministic
- In short a design frequently
  - Is theoretically correct
  - Simulates OK
  - Fails in a real part



Why are they important

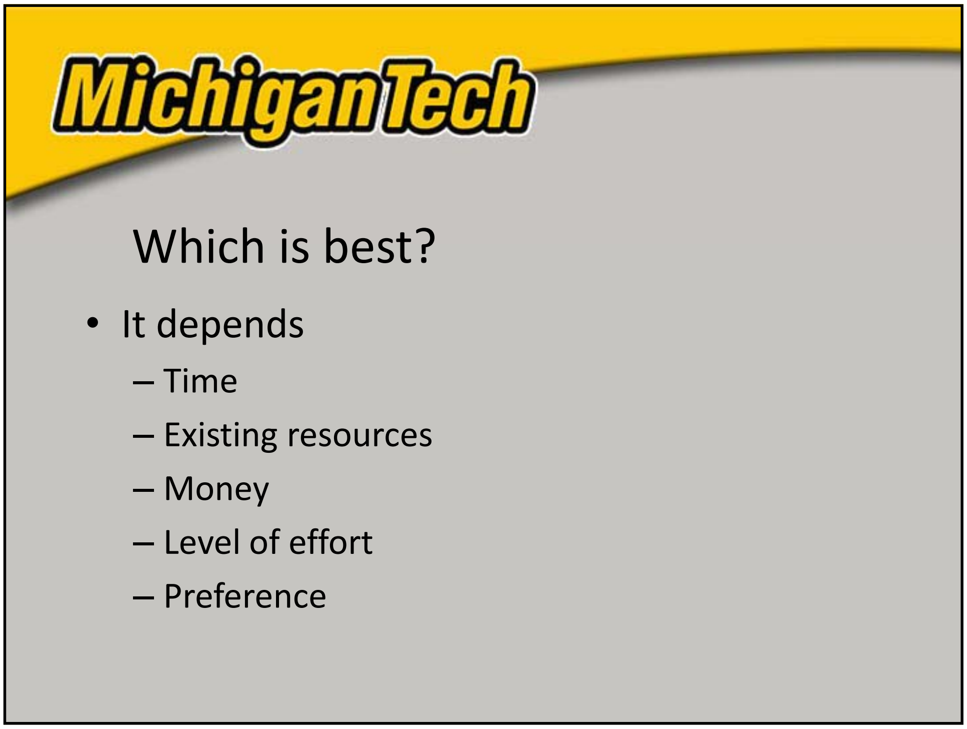
- They have the ability to revolutionize the way that prototyping is done.
- Allows companies to get to market quicker and stay in market longer.

Xilinx

- Largest manufacturer of HW
- Develop hardware and software
- Embedded PowerPC
- University Program

Altera

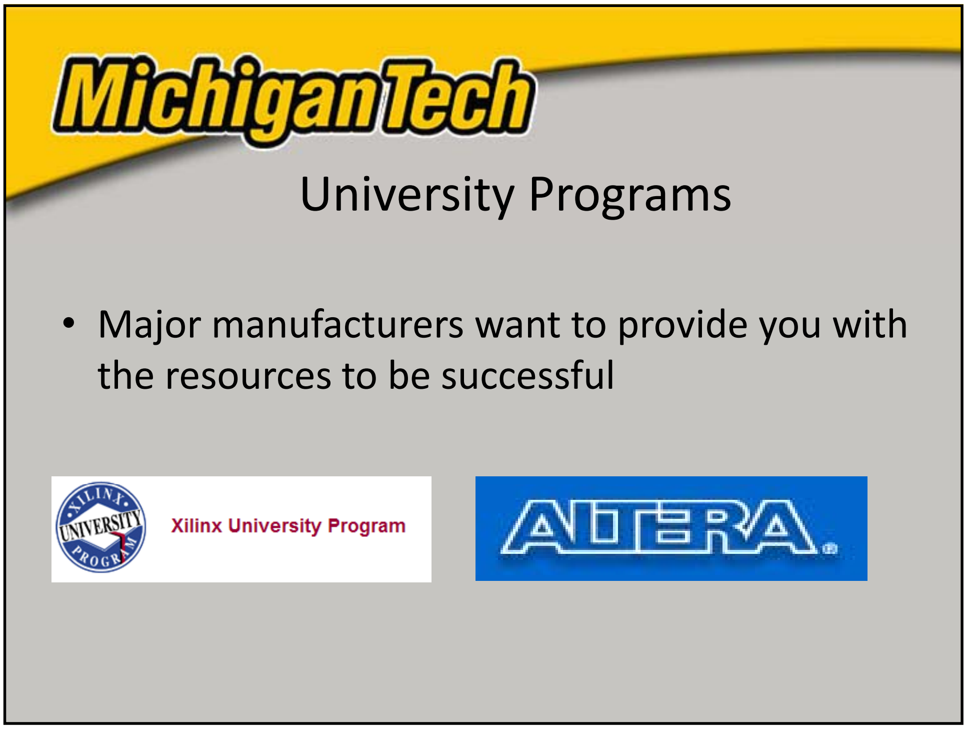
- Second largest manufacturer
- Develop HW and SW
- University Program



MichiganTech

Which is best?

- It depends
  - Time
  - Existing resources
  - Money
  - Level of effort
  - Preference



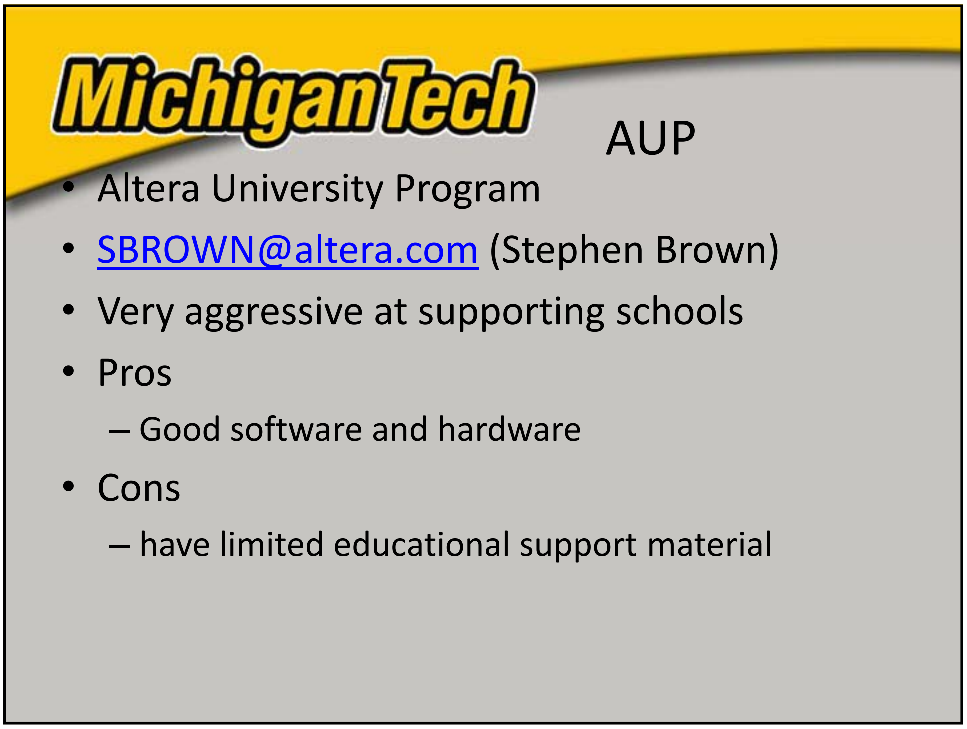
MichiganTech

University Programs

- Major manufacturers want to provide you with the resources to be successful

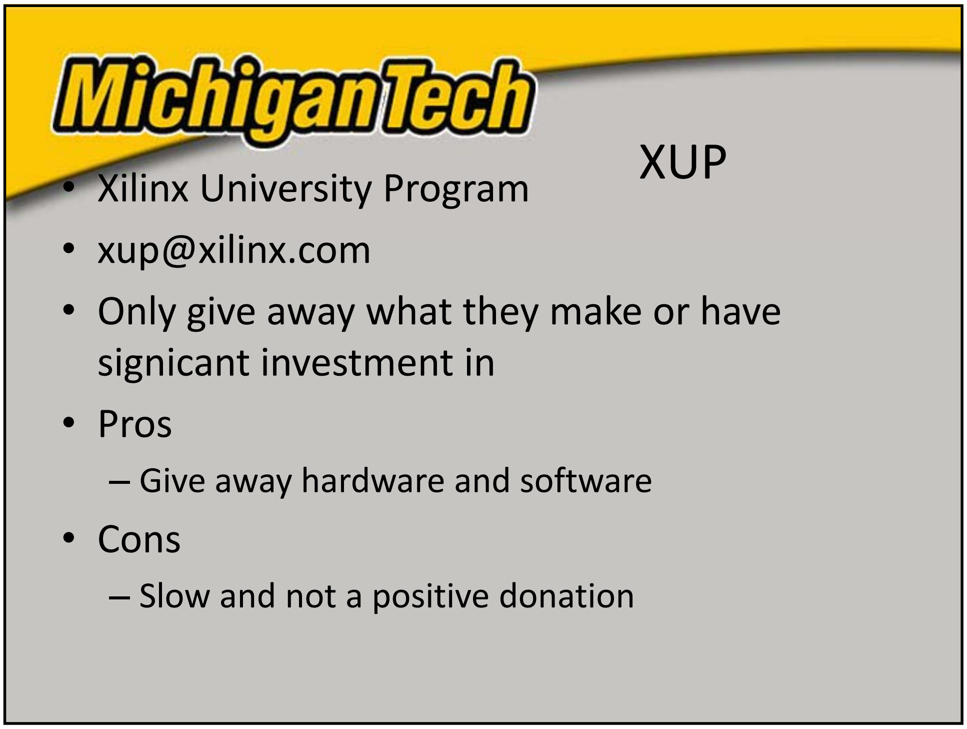
XILINX UNIVERSITY PROGRAM Xilinx University Program

ALTERA.



AUP

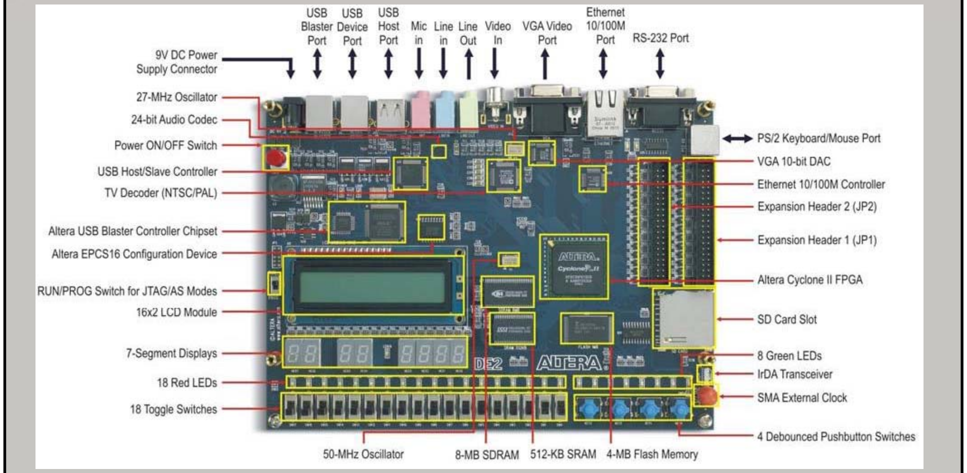
- Altera University Program
- SBROWN@altera.com (Stephen Brown)
- Very aggressive at supporting schools
- Pros
  - Good software and hardware
- Cons
  - have limited educational support material



XUP

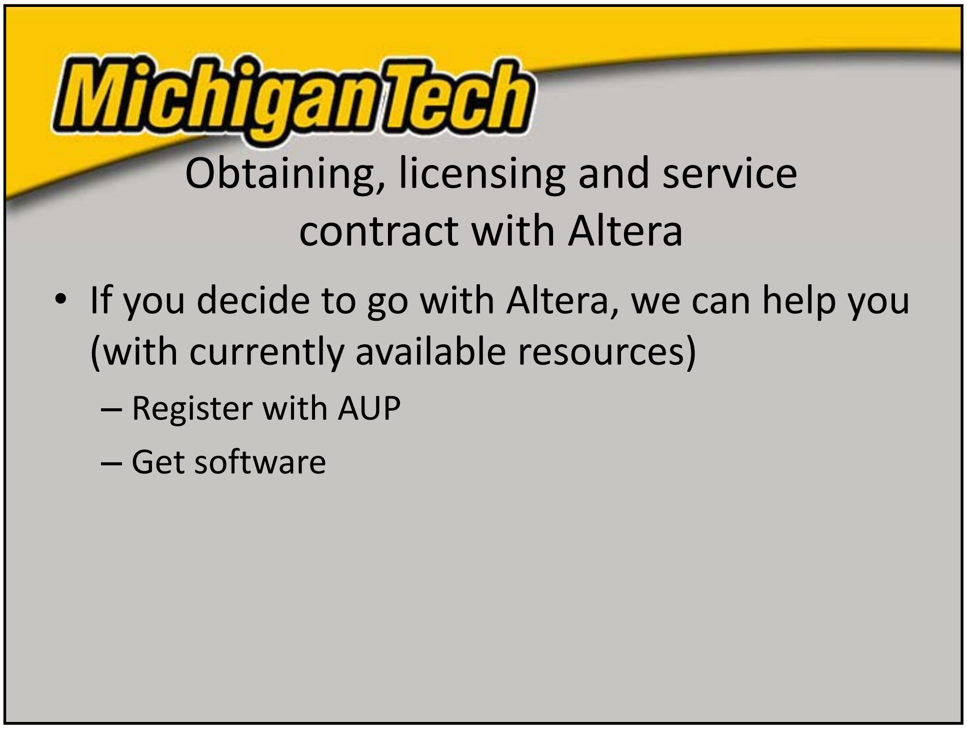
- Xilinx University Program
- xup@xilinx.com
- Only give away what they make or have signicant investment in
- Pros
  - Give away hardware and software
- Cons
  - Slow and not a positive donation

Altera DE2 Development Board



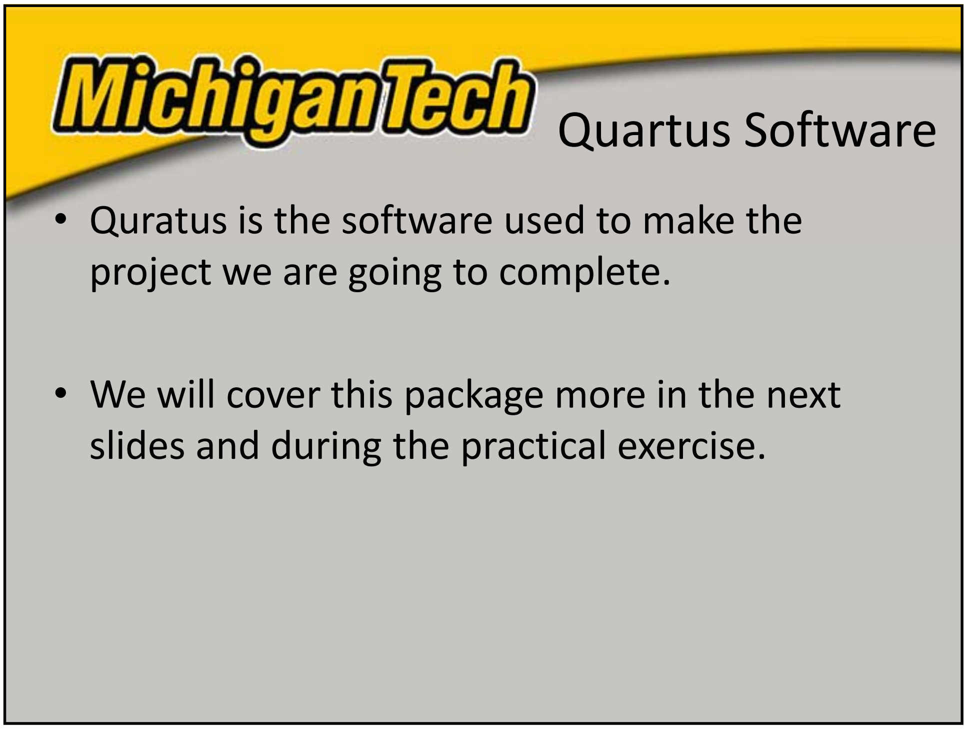
What do you really need to start?

- Quartus Software
- DE2 FPGA board



Obtaining, licensing and service contract with Altera

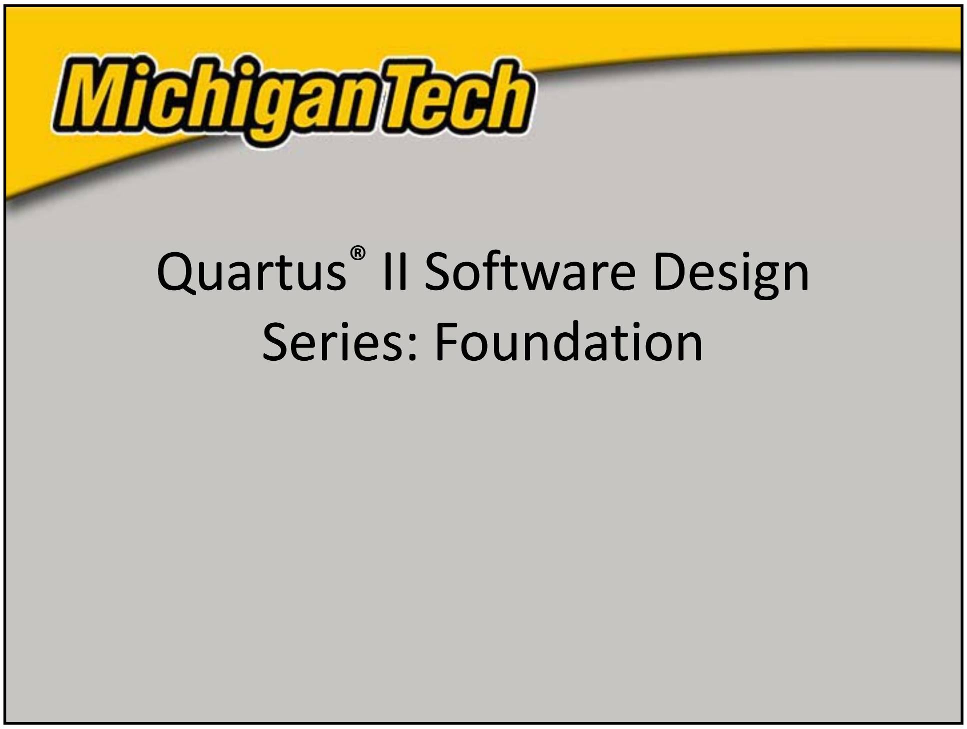
- If you decide to go with Altera, we can help you (with currently available resources)
  - Register with AUP
  - Get software



Quartus Software

- Quratus is the software used to make the project we are going to complete.

- We will cover this package more in the next slides and during the practical exercise.

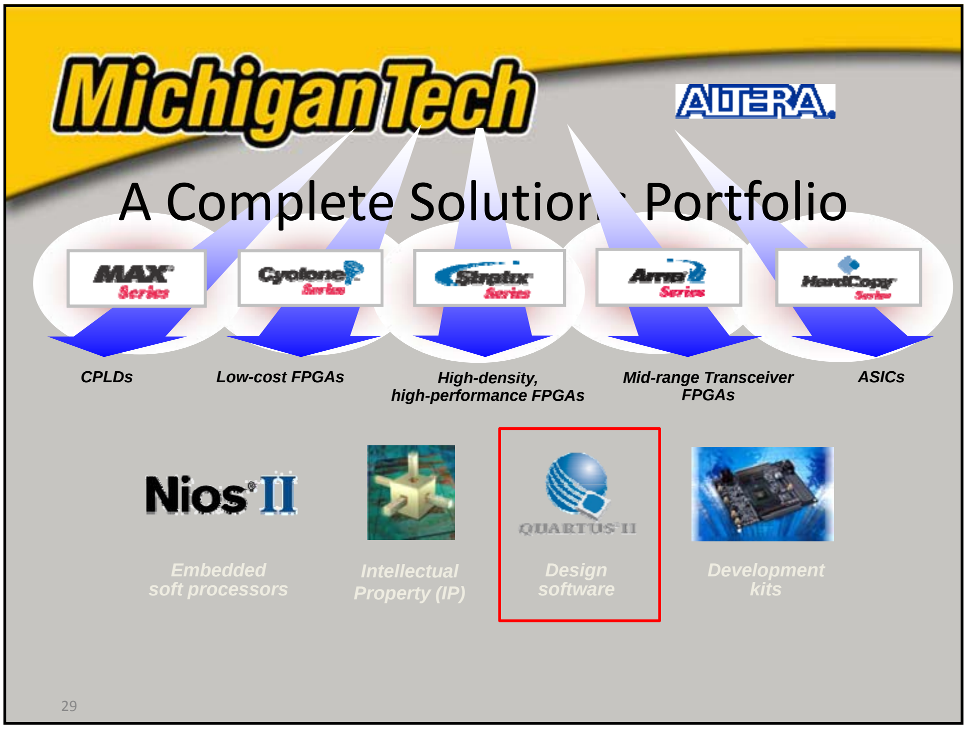


Quartus® II Software Design Series: Foundation

Objectives

- Create a new Quartus® II project
- Choose supported design entry methods
- Compile a design into an FPGA
- Locate resulting compilation information
- Create design constraints (assignments & settings)
- Manage I/O assignments
- Perform timing analysis & obtain results

28


MichiganTech
ALTERA
MAX Series
Cyclone Series
Stratix Series
Arria II Series
HardCopy Series
Nios II
QUARTUS II

A Complete Solution Portfolio

CPLDs

Low-cost FPGAs

High-density, high-performance FPGAs

Mid-range Transceiver FPGAs

ASICs

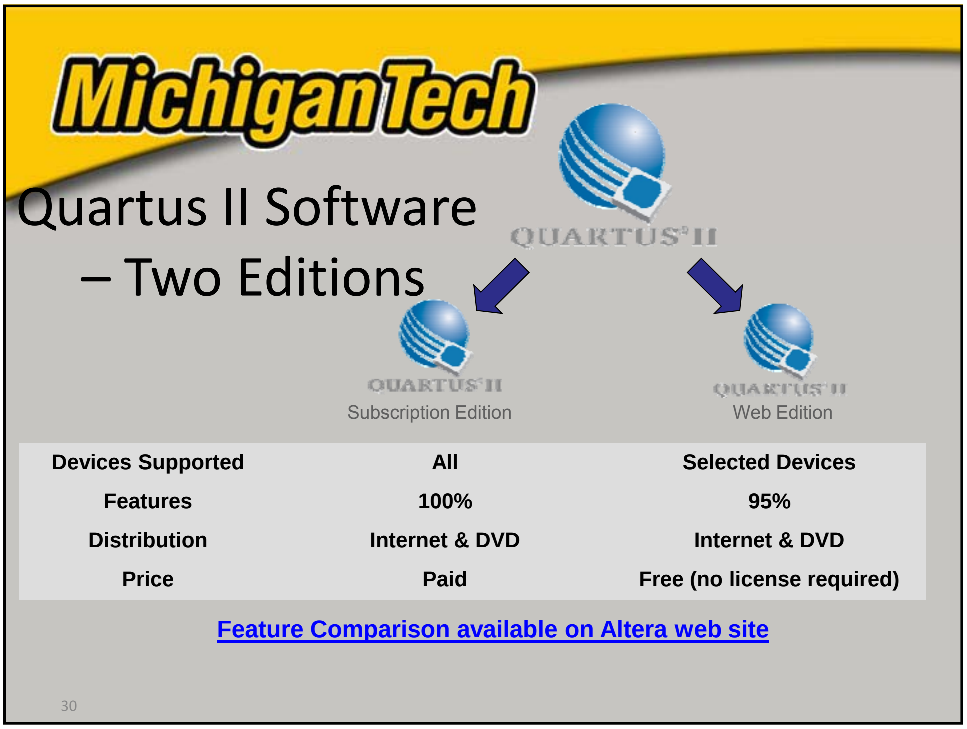
Embedded soft processors

Intellectual Property (IP)

Design software

Development kits

29

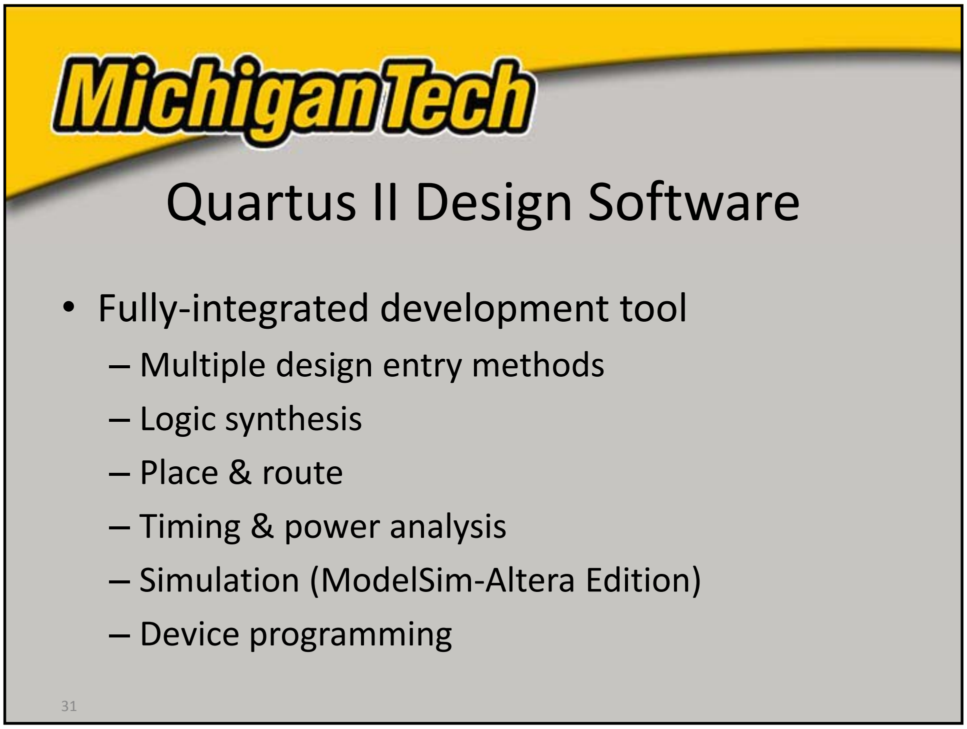

MichiganTech
QUARTUS II
QUARTUS II
QUARTUS II

Quartus II Software – Two Editions

	Subscription Edition	Web Edition
Devices Supported	All	Selected Devices
Features	100%	95%
Distribution	Internet & DVD	Internet & DVD
Price	Paid	Free (no license required)

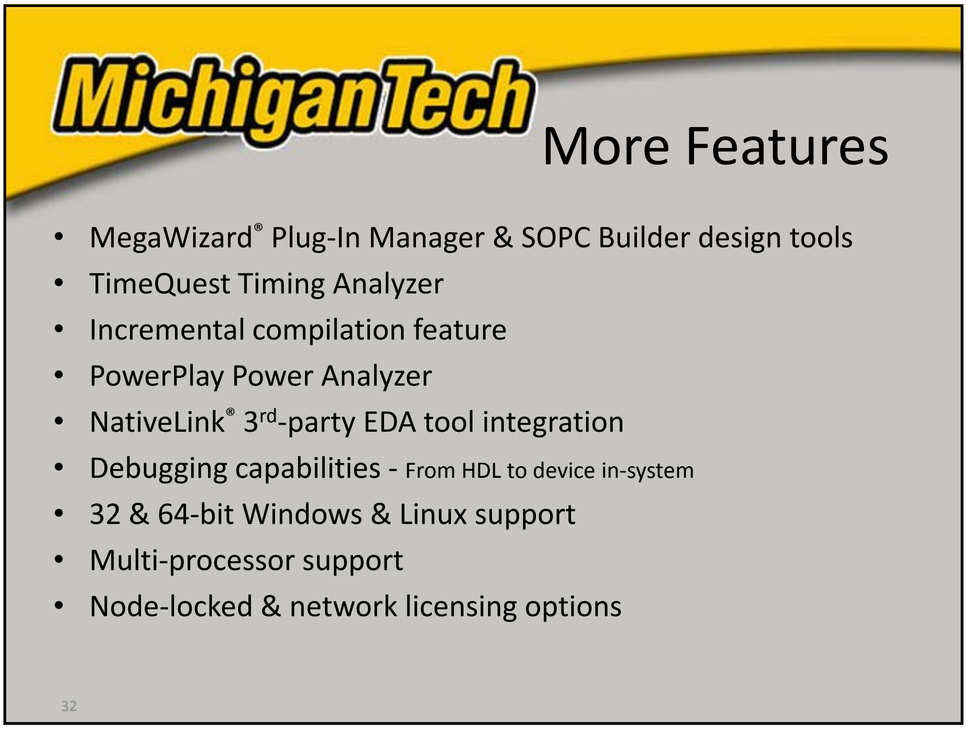
Feature Comparison available on Altera web site

30



Quartus II Design Software

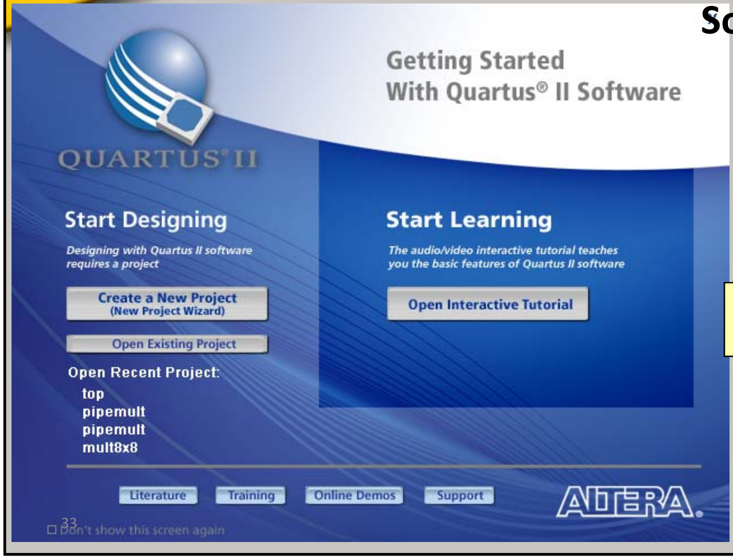
- Fully-integrated development tool
  - Multiple design entry methods
  - Logic synthesis
  - Place & route
  - Timing & power analysis
  - Simulation (ModelSim-Altera Edition)
  - Device programming



More Features

- MegaWizard® Plug-In Manager & SOPC Builder design tools
- TimeQuest Timing Analyzer
- Incremental compilation feature
- PowerPlay Power Analyzer
- NativeLink® 3rd-party EDA tool integration
- Debugging capabilities - From HDL to device in-system
- 32 & 64-bit Windows & Linux support
- Multi-processor support
- Node-locked & network licensing options

Welcome to the Quartus II Software!



Getting Started With Quartus® II Software

Start Designing

Designing with Quartus II software requires a project

Start Learning

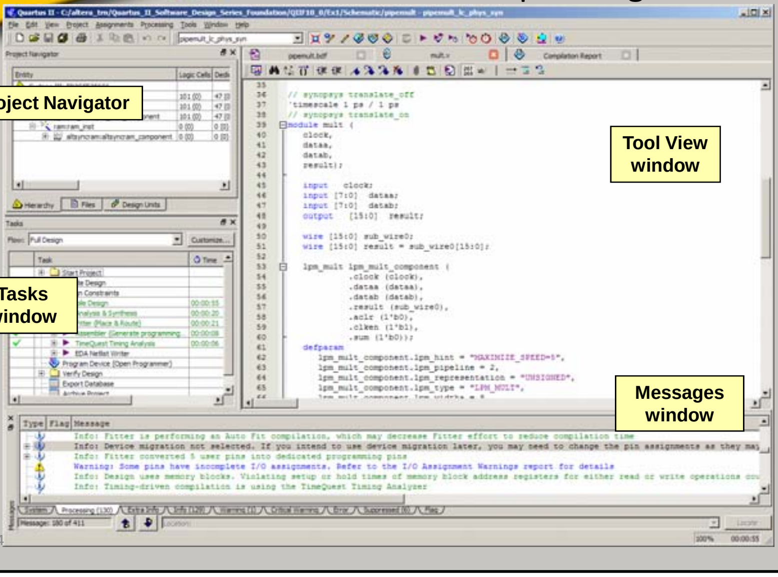
The audio/video interactive tutorial teaches you the basic features of Quartus II software

Open Recent Project:
top
pipemult
pipemult
mult8x8

QUARTUS® II
Create a New Project (New Project Wizard)
Open Existing Project
Open Interactive Tutorial
Literature
Training
Online Demos
Support
ALTERA.
□ Don't show this screen again

Turn on or off in Tools ⇒ Options

Quartus II Default Operating Environment



```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module mult (
    clock,
    dataa,
    datab,
    result);

    input   clock;
    input [7:0]  dataa;
    input [7:0]  datab;
    output   [15:0]  result;

    wire [15:0] sub_wire0;
    wire [15:0] result = sub_wire0[15:0];

    lpm_mult lpm_mult_component (
                .clock (clock),
                .dataa (dataa),
                .datab (datab),
                .result (sub_wire0),
                .aclr (1'b0),
                .clken (1'b1),
                .sum (1'b0));
    defparam
        lpm_mult_component.lpm_hint = "MAXIMIZE_SPEED=5",
        lpm_mult_component.lpm_pipeline = 2,
        lpm_mult_component.lpm_representation = "UNSIGNED",
        lpm_mult_component.lpm_type = "LPM_MULT",
```

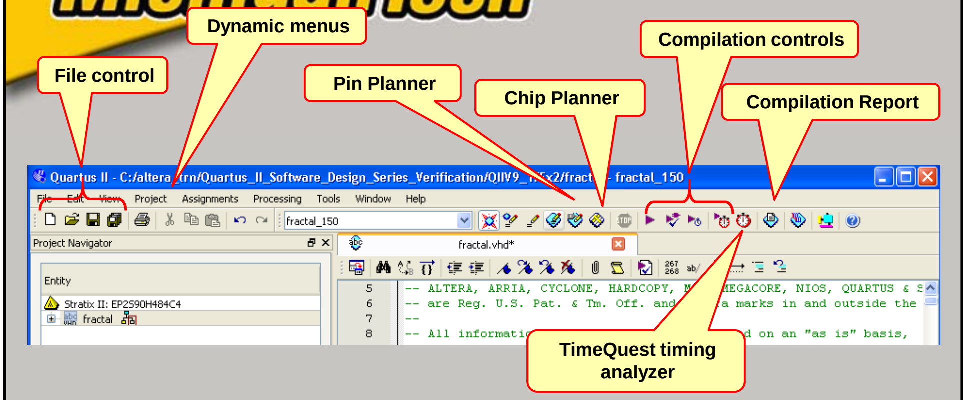
```
Type Flag Message
Info: Fitter is performing an Auto Fit compilation, which may decrease Fitter effort to reduce compilation time
Info: Device migration not selected. If you intend to use device migration later, you may need to change the pin assignments as they may
Info: Fitter converted 5 user pins into dedicated programming pins
Warning: Some pins have incomplete I/O assignments. Refer to the I/O Assignment Warnings report for details
Info: Design uses memory blocks. Violating setup or hold times of memory block address registers for either read or write operations cou
Info: Timing-driven compilation is using the TimeQuest Timing Analyzer
```

Project Navigator

Tasks window

Tool View window

Messages window


Dynamic menus
File control
Pin Planner
Chip Planner
Compilation controls
Compilation Report
TimeQuest timing analyzer
Quartus II - C:/altera_trn/Quartus_II_Software_Design_Series_Verification/QIIV9_1/Ex2/fract - fractal_150
File Edit View Project Assignments Processing Tools Window Help
fractal_150
Project Navigator
fractal.vhd*
Entity
Stratix II: EP2S90H484C4
fractal

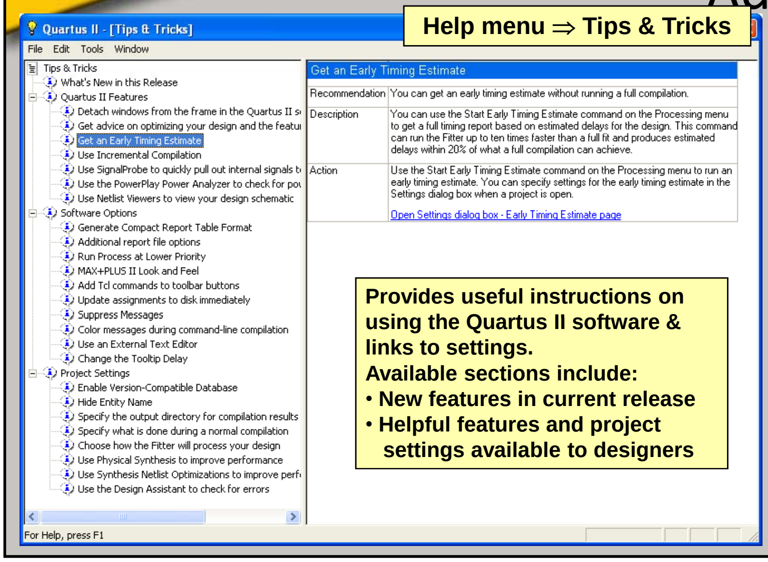
```
-- ALTERA, ARRIA, CYCLONE, HARDCOPY, M    MEGACORE, NIOS, QUARTUS & S
-- are Reg. U.S. Pat. & Tm. Off. and     a marks in and outside the
--
-- All informati                         d on an "as is" basis,
```

Main Toolbar

To reset views:
1. **Tools ⇒ Customize ⇒ Toolbars ⇒ Reset All**
2. **Restart Quartus II**

35

Tips & Tricks Advisor

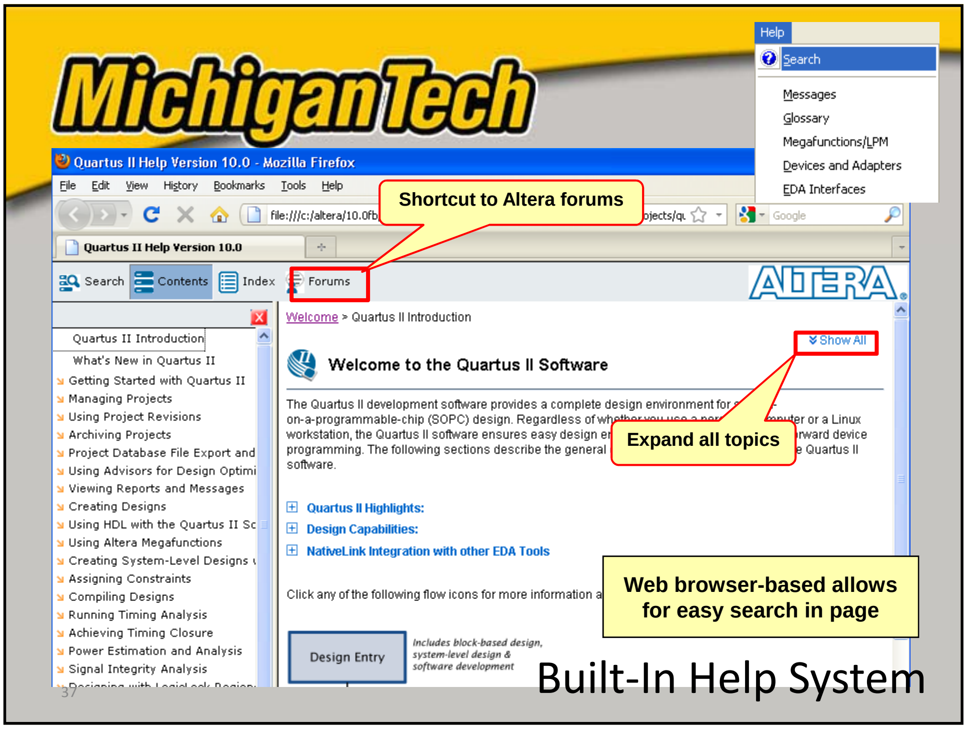

Help menu ⇒ Tips & Tricks

- Tips & Tricks
  - What's New in this Release
  - Quartus II Features
    - Detach windows from the frame in the Quartus II s
    - Get advice on optimizing your design and the featu
    - Get an Early Timing Estimate
    - Use Incremental Compilation
    - Use SignalProbe to quickly pull out internal signals t
    - Use the PowerPlay Power Analyzer to check for po
    - Use Netlist Viewers to view your design schematic
  - Software Options
    - Generate Compact Report Table Format
    - Additional report file options
    - Run Process at Lower Priority
    - MAX+PLUS II Look and Feel
    - Add Tcl commands to toolbar buttons
    - Update assignments to disk immediately
    - Suppress Messages
    - Color messages during command-line compilation
    - Use an External Text Editor
    - Change the Tooltip Delay
  - Project Settings
    - Enable Version-Compatible Database
    - Hide Entity Name
    - Specify the output directory for compilation results
    - Specify what is done during a normal compilation
    - Choose how the Fitter will process your design
    - Use Physical Synthesis to improve performance
    - Use Synthesis Netlist Optimizations to improve perf
    - Use the Design Assistant to check for errors

Get an Early Timing Estimate	
Recommendation	You can get an early timing estimate without running a full compilation.
Description	You can use the Start Early Timing Estimate command on the Processing menu to get a full timing report based on estimated delays for the design. This command can run the Fitter up to ten times faster than a full fit and produces estimated delays within 20% of what a full compilation can achieve.
Action	Use the Start Early Timing Estimate command on the Processing menu to run an early timing estimate. You can specify settings for the early timing estimate in the Settings dialog box when a project is open. Open Settings dialog box - Early Timing Estimate page

For Help, press F1

Provides useful instructions on using the Quartus II software & links to settings.
Available sections include:
- **New features in current release**
- **Helpful features and project settings available to designers**


MichiganTech
Help
Search
Messages
Glossary
Megafunctions/LPM
Devices and Adapters
EDA Interfaces
Quartus II Help Version 10.0 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///c:/altera/10.0fb
ojects/qu
Google
Quartus II Help Version 10.0
Search
Contents
Index
Forums
ALTERA
Quartus II Introduction
What's New in Quartus II
Getting Started with Quartus II
Managing Projects
Using Project Revisions
Archiving Projects
Project Database File Export and
Using Advisors for Design Optimi
Viewing Reports and Messages
Creating Designs
Using HDL with the Quartus II Sc
Using Altera Megafunctions
Creating System-Level Designs
Assigning Constraints
Compiling Designs
Running Timing Analysis
Achieving Timing Closure
Power Estimation and Analysis
Signal Integrity Analysis
Welcome > Quartus II Introduction
Show All
Welcome to the Quartus II Software
Quartus II Highlights:
Design Capabilities:
NativeLink Integration with other EDA Tools
Design Entry
Includes block-based design, system-level design & software development
37

The Quartus II development software provides a complete design environment for s… on-a-programmable-chip (SOPC) design. Regardless of whether you use a pers… …mputer or a Linux workstation, the Quartus II software ensures easy design en… …rward device programming. The following sections describe the general … …e Quartus II software.

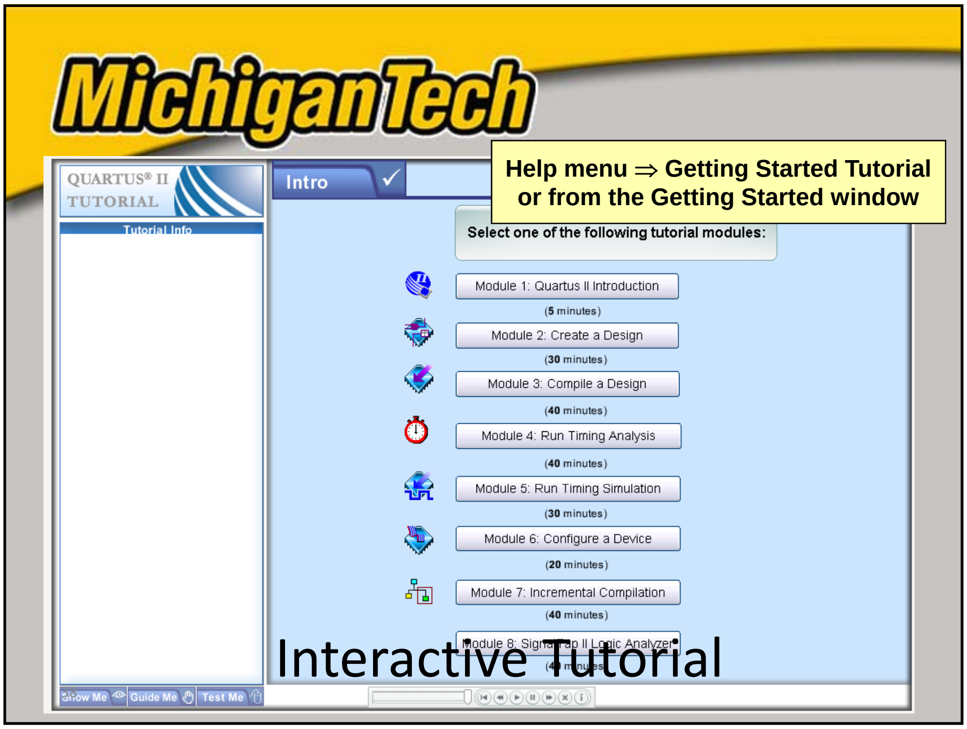
Click any of the following flow icons for more information a…

Shortcut to Altera forums

Expand all topics

Web browser-based allows for easy search in page

Built-In Help System



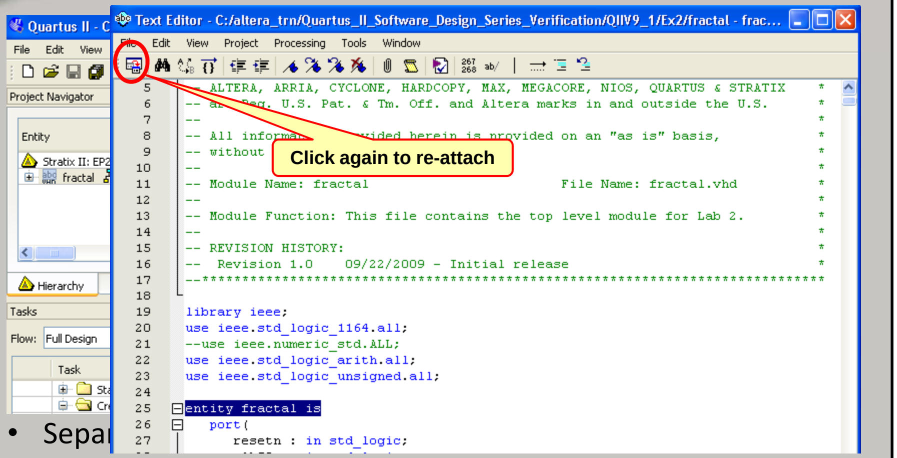
Help menu ⇒ Getting Started Tutorial or from the Getting Started window

Select one of the following tutorial modules:

- Module 1: Quartus II Introduction (**5** minutes)
- Module 2: Create a Design (**30** minutes)
- Module 3: Compile a Design (**40** minutes)
- Module 4: Run Timing Analysis (**40** minutes)
- Module 5: Run Timing Simulation (**30** minutes)
- Module 6: Configure a Device (**20** minutes)
- Module 7: Incremental Compilation (**40** minutes)
- Module 8: SignalTap II Logic Analyzer (**40** minutes)

Interactive Tutorial

Detachable Windows


Quartus II - C
File Edit View
Project Navigator
Entity
Stratix II: EP2
fractal
Hierarchy
Tasks
Flow: Full Design
Task
Sta
Cre
Text Editor - C:/altera_trn/Quartus_II_Software_Design_Series_Verification/QIIV9_1/Ex2/fractal - frac...
File Edit View Project Processing Tools Window

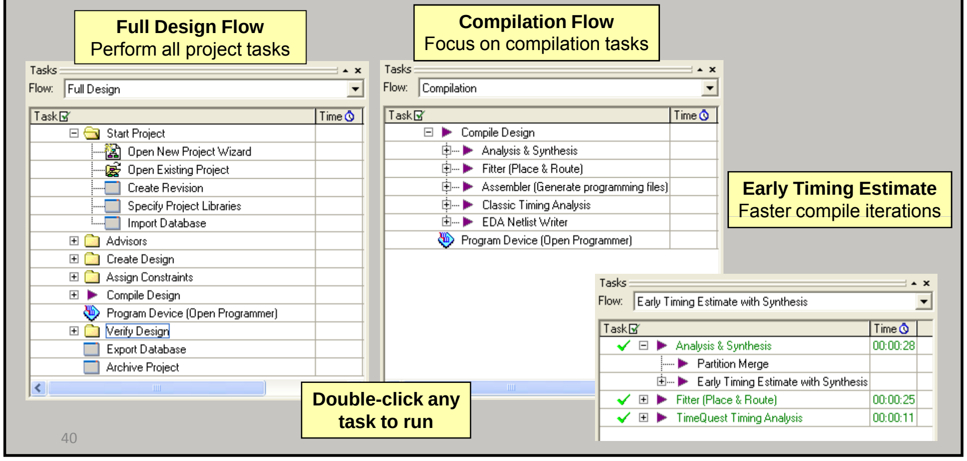
```
-- ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX     *
-- ar   Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.      *
--                                                                             *
-- All informa          vided herein is provided on an "as is" basis,          *
-- without                                                                     *
--                                                                             *
-- Module Name: fractal                            File Name: fractal.vhd      *
--                                                                             *
-- Module Function: This file contains the top level module for Lab 2.         *
--                                                                             *
-- REVISION HISTORY:                                                           *
--  Revision 1.0    09/22/2009 - Initial release                               *
--*****************************************************************************

library ieee;
use ieee.std_logic_1164.all;
--use ieee.numeric_std.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fractal is
   port(
      resetn : in std_logic;
```

Click again to re-attach

- Separ
(**Window** menu ⇒ **Detach/Attach Window**)

Tasks Window

- Easy access to most Quartus II functions
- Organized into related tasks within three task flows



Full Design Flow
Perform all project tasks

Tasks
Flow: Full Design

Task	Time
Start Project	
Open New Project Wizard	
Open Existing Project	
Create Revision	
Specify Project Libraries	
Import Database	
Advisors	
Create Design	
Assign Constraints	
Compile Design	
Program Device (Open Programmer)	
Verify Design	
Export Database	
Archive Project	

Compilation Flow
Focus on compilation tasks

Tasks
Flow: Compilation

Task	Time
Compile Design	
Analysis & Synthesis	
Fitter (Place & Route)	
Assembler (Generate programming files)	
Classic Timing Analysis	
EDA Netlist Writer	
Program Device (Open Programmer)	

Early Timing Estimate
Faster compile iterations

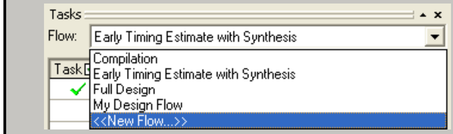
Tasks
Flow: Early Timing Estimate with Synthesis

Task	Time
✓ Analysis & Synthesis	00:00:28
Partition Merge	
Early Timing Estimate with Synthesis	
✓ Fitter (Place & Route)	00:00:25
✓ TimeQuest Timing Analysis	00:00:11

Double-click any task to run

Custom Task Flow

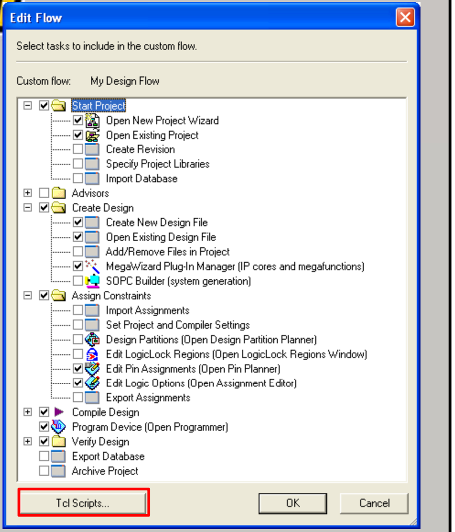
- Customize the Tasks display
- Add Tcl scripts for quick access



Tasks

Flow: Early Timing Estimate with Synthesis

- Compilation
- Early Timing Estimate with Synthesis
- Full Design
- My Design Flow
- <<New Flow...>>



Select tasks to include in the custom flow.

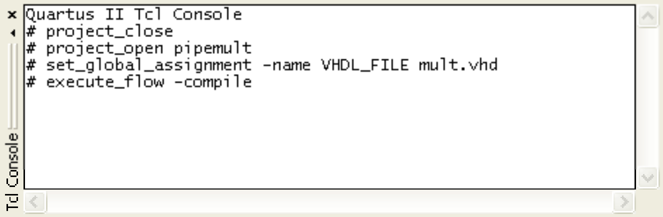
Custom flow: My Design Flow

- ☑ Start Project
  - ☑ Open New Project Wizard
  - ☑ Open Existing Project
  - □ Create Revision
  - □ Specify Project Libraries
  - □ Import Database
- □ Advisors
- ☑ Create Design
  - ☑ Create New Design File
  - ☑ Open Existing Design File
  - □ Add/Remove Files in Project
  - ☑ MegaWizard Plug-In Manager (IP cores and megafunctions)
  - □ SOPC Builder (system generation)
- ☑ Assign Constraints
  - □ Import Assignments
  - □ Set Project and Compiler Settings
  - □ Design Partitions (Open Design Partition Planner)
  - □ Edit LogicLock Regions (Open LogicLock Regions Window)
  - ☑ Edit Pin Assignments (Open Pin Planner)
  - ☑ Edit Logic Options (Open Assignment Editor)
  - □ Export Assignments
- ☑ Compile Design
- ☑ Program Device (Open Programmer)
- ☑ Verify Design
- □ Export Database
- □ Archive Project

Tcl Console Window

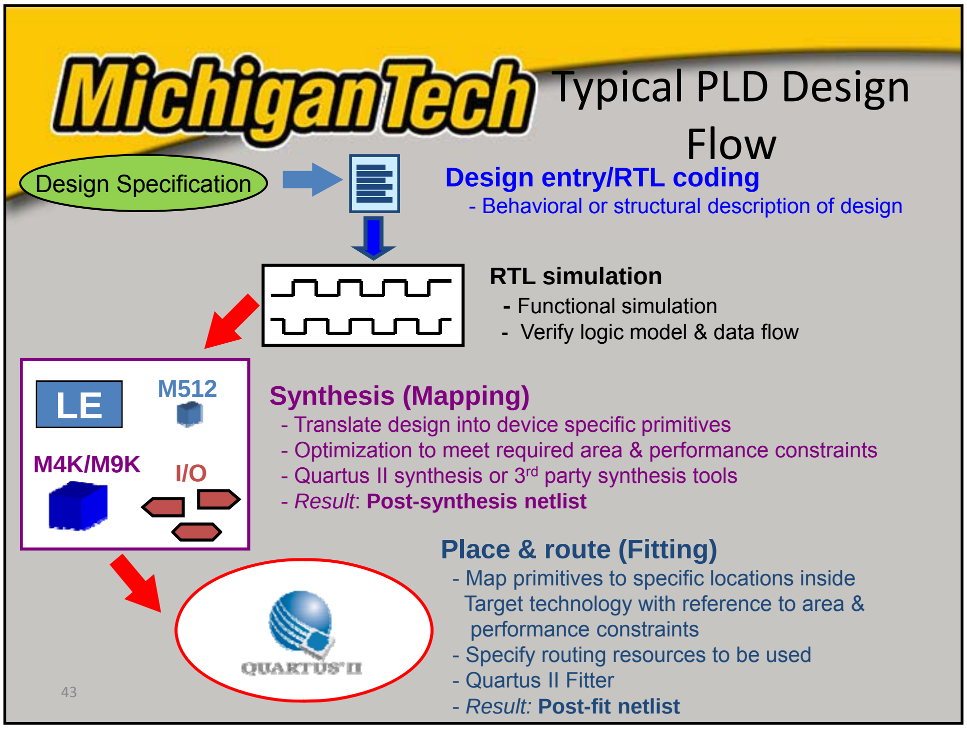
- Enter and execute Tcl commands directly in the GUI

View menu ⇒ Utility Windows ⇒ Tcl Console



```
Quartus II Tcl Console
# project_close
# project_open pipemult
# set_global_assignment -name VHDL_FILE mult.vhd
# execute_flow -compile
```

- Execute from command-line using Tcl shell
  - `quartus_sh -shell`
- Run complete scripts from **Tools** menu ⇒ **Tcl Scripts**



Typical PLD Design Flow

Design Specification

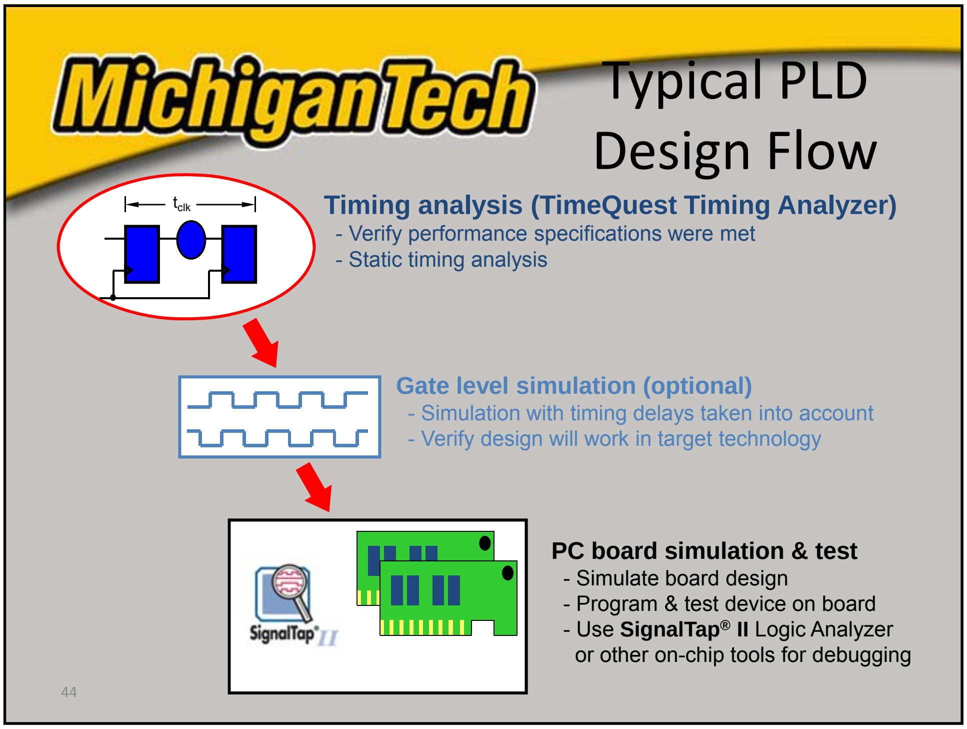
Design entry/RTL coding
- Behavioral or structural description of design

RTL simulation
- Functional simulation
- Verify logic model & data flow

Synthesis (Mapping)
- Translate design into device specific primitives
- Optimization to meet required area & performance constraints
- Quartus II synthesis or 3rd party synthesis tools
- *Result*: **Post-synthesis netlist**

Place & route (Fitting)
- Map primitives to specific locations inside Target technology with reference to area & performance constraints
- Specify routing resources to be used
- Quartus II Fitter
- *Result:* **Post-fit netlist**

43



Typical PLD Design Flow

Timing analysis (TimeQuest Timing Analyzer)
- Verify performance specifications were met
- Static timing analysis

Gate level simulation (optional)
- Simulation with timing delays taken into account
- Verify design will work in target technology

PC board simulation & test
- Simulate board design
- Program & test device on board
- Use **SignalTap® II** Logic Analyzer or other on-chip tools for debugging

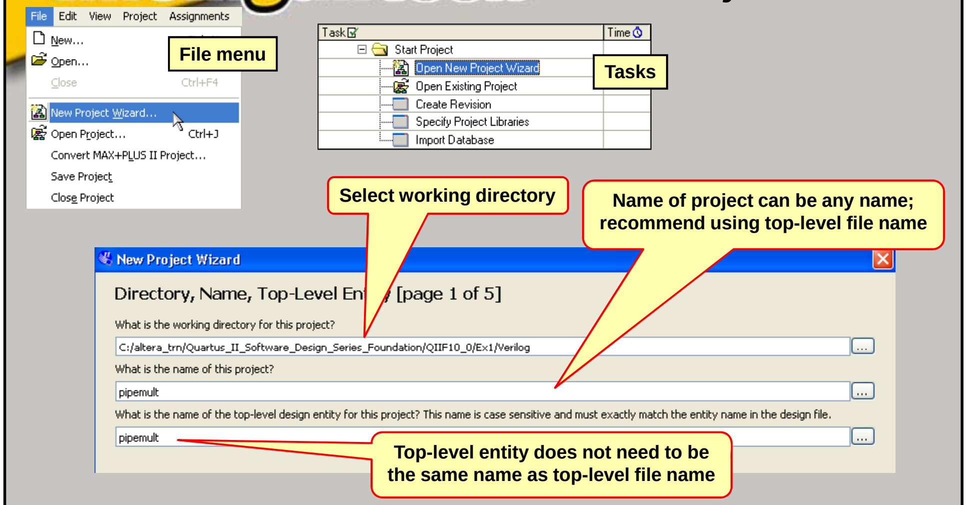
44

Quartus II Projects

- Description
  - Collection of related design files & libraries
  - Must have a designated top-level entity
  - Target a single device
  - Store settings in Quartus II Settings File (.QSF)
  - Compiled netlist information stored in **db** folder in project directory
- Create new projects with New Project Wizard
  - Can be created using Tcl scripts

45

New Project Wizard



File menu

Tasks

Select working directory

Name of project can be any name; recommend using top-level file name

Directory, Name, Top-Level Ent[y [page 1 of 5]

What is the working directory for this project?

C:/altera_trn/Quartus_II_Software_Design_Series_Foundation/QIIF10_0/Ex1/Verilog

What is the name of this project?

pipemult

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

pipemult

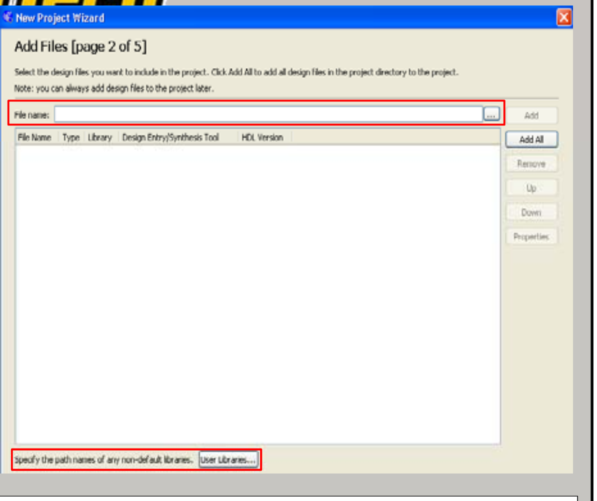
Top-level entity does not need to be the same name as top-level file name

Tcl: project_new <project_name>

46

Add Files

- Add design files
  - Graphic
  - VHDL
  - Verilog
  - SystemVerilog
  - EDIF
  - VQM
- Add library paths
  - User libraries
  - MegaCore®/AMPP℠
  - Pre-compiled VHDL packages



New Project Wizard

Add Files [page 2 of 5]

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name:

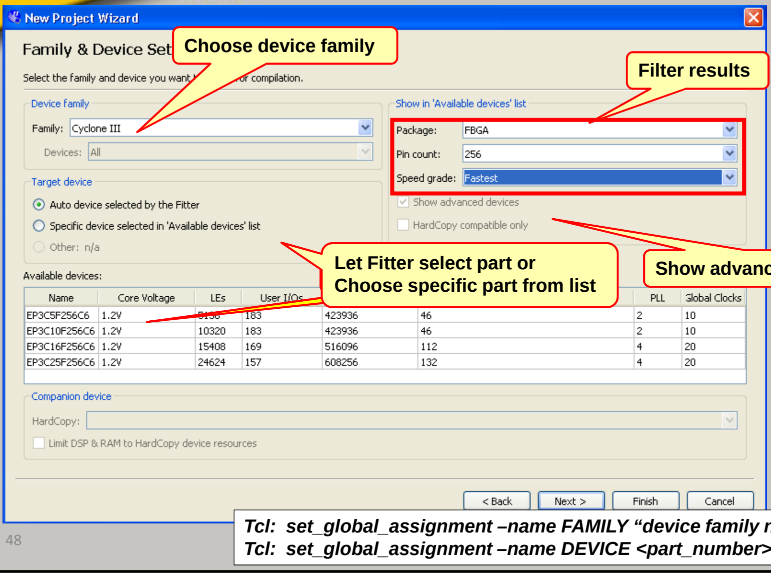
File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version

Specify the path names of any non-default libraries.

Tcl: set_global_assignment –name VHDL_FILE <filename.vhd>
Tcl: set_global_assignment –name USER_LIBRARIES <library_path_name>

47

Device Selection



New Project Wizard

Family & Device Settings

Select the family and device you want to target for compilation.

Device family

Family: Cyclone III

Devices: All

Target device

● Auto device selected by the Fitter

○ Specific device selected in 'Available devices' list

○ Other: n/a

Show in 'Available devices' list

Package: FBGA

Pin count: 256

Speed grade: Fastest

☑ Show advanced devices

□ HardCopy compatible only

Available devices:

Name	Core Voltage	LEs	User I/Os			PLL	Global Clocks
EP3C5F256C6	1.2V	5136	183	423936	46	2	10
EP3C10F256C6	1.2V	10320	183	423936	46	2	10
EP3C16F256C6	1.2V	15408	169	516096	112	4	20
EP3C25F256C6	1.2V	24624	157	608256	132	4	20

Companion device

HardCopy:

□ Limit DSP & RAM to HardCopy device resources

Choose device family

Filter results

Let Fitter select part or Choose specific part from list

Show advanced devices

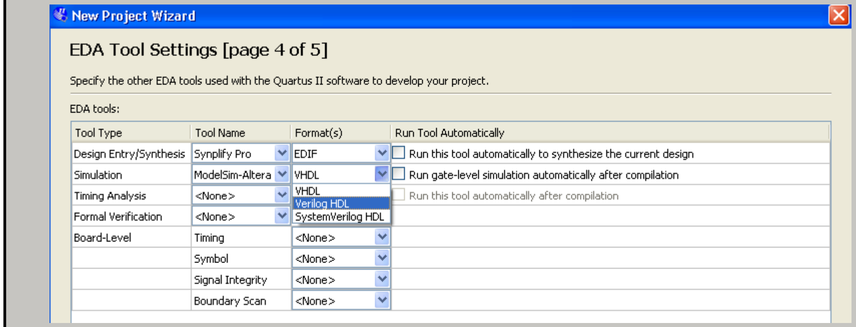
Tcl: set_global_assignment –name FAMILY "device family name"
Tcl: set_global_assignment –name DEVICE <part_number>

48

MichiganTech

EDA Tool Settings

- Choose EDA tools and file formats
- Settings can be changed or added later


New Project Wizard

EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

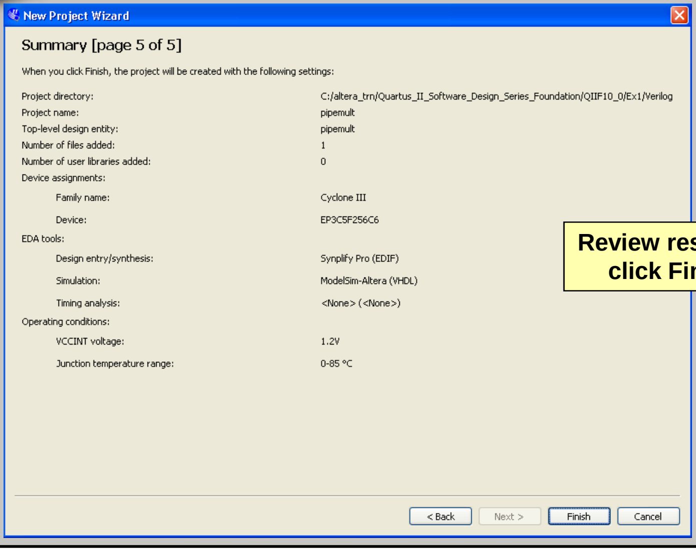
EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	Synplify Pro	EDIF	□ Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	□ Run gate-level simulation automatically after compilation
Timing Analysis	<None>		□ Run this tool automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

See handbook for Tcl command format

MichiganTech

Done!


New Project Wizard

Summary [page 5 of 5]

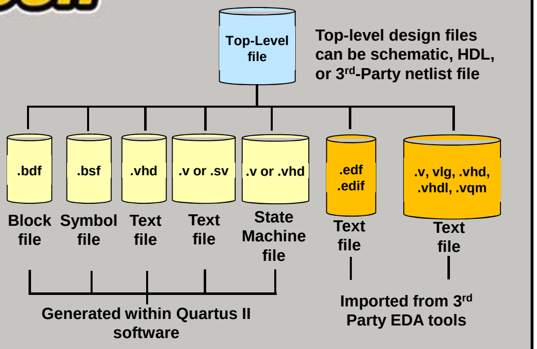
When you click Finish, the project will be created with the following settings:

Project directory:	C:/altera_trn/Quartus_II_Software_Design_Series_Foundation/QIIF10_0/Ex1/Verilog
Project name:	pipemult
Top-level design entity:	pipemult
Number of files added:	1
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone III
Device:	EP3C5F256C6
EDA tools:	
Design entry/synthesis:	Synplify Pro (EDIF)
Simulation:	ModelSim-Altera (VHDL)
Timing analysis:	<None> (<None>)
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 °C

Review results & click Finish

Design Entry Methods

- Quartus II design entry
  - Text editor
    - VHDL
    - Verilog or SystemVerilog
  - Schematic editor
    - Block Diagram File
  - State machine editor
    - HDL from state machine file
  - Memory editor
    - HEX
    - MIF
- 3rd-party EDA tools
  - EDIF 2 0 0
  - Verilog Quartus Mapping (.VQM)
- Mixing & matching design files allowed



Top-Level file

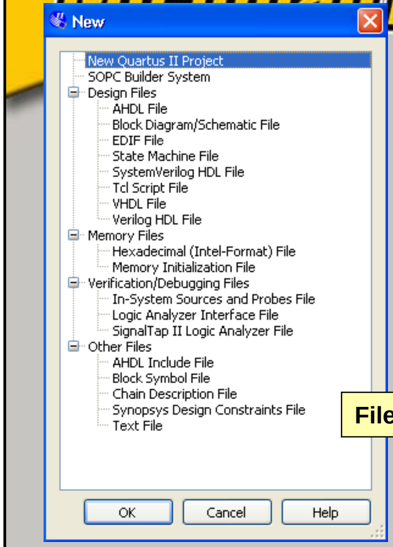
Top-level design files can be schematic, HDL, or 3rd-Party netlist file

.bdf	.bsf	.vhd	.v or .sv	.v or .vhd	.edf .edif	.v, vlg, .vhd, .vhdl, .vqm
Block file	Symbol file	Text file	Text file	State Machine file	Text file	Text file

Generated within Quartus II software

Imported from 3rd Party EDA tools

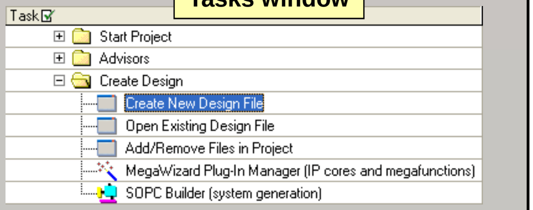
Creating New Design Files (& Others)



New

- New Quartus II Project
- SOPC Builder System
- Design Files
  - AHDL File
  - Block Diagram/Schematic File
  - EDIF File
  - State Machine File
  - SystemVerilog HDL File
  - Tcl Script File
  - VHDL File
  - Verilog HDL File
- Memory Files
  - Hexadecimal (Intel-Format) File
  - Memory Initialization File
- Verification/Debugging Files
  - In-System Sources and Probes File
  - Logic Analyzer Interface File
  - SignalTap II Logic Analyzer File
- Other Files
  - AHDL Include File
  - Block Symbol File
  - Chain Description File
  - Synopsys Design Constraints File
  - Text File

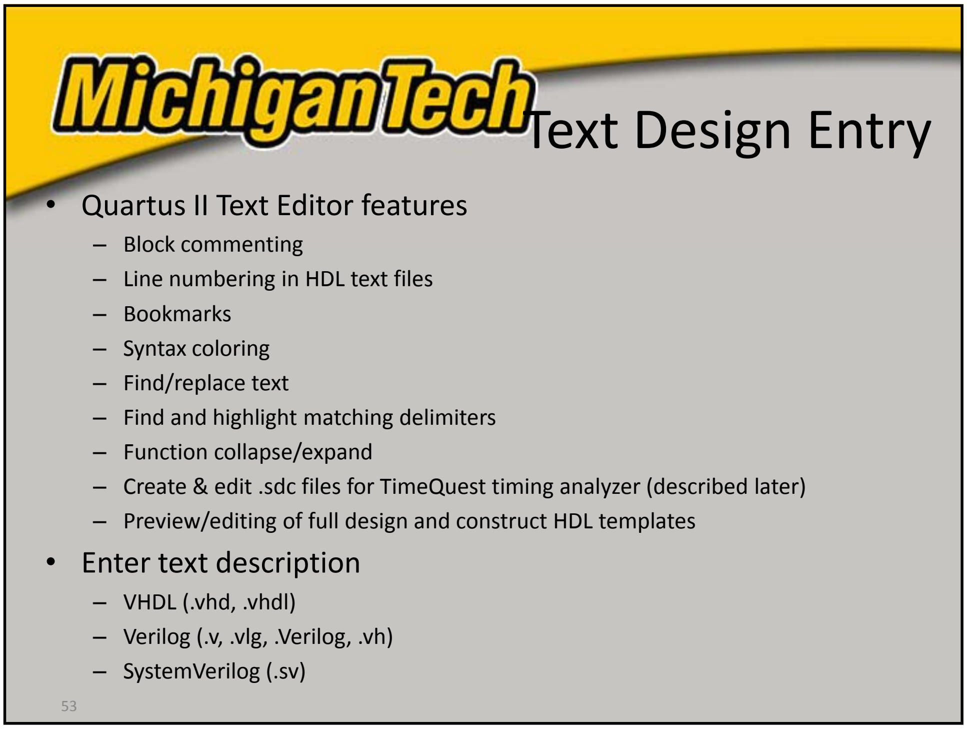
Tasks window



Task

- Start Project
- Advisors
- Create Design
  - Create New Design File
  - Open Existing Design File
  - Add/Remove Files in Project
  - MegaWizard Plug-In Manager (IP cores and megafunctions)
  - SOPC Builder (system generation)

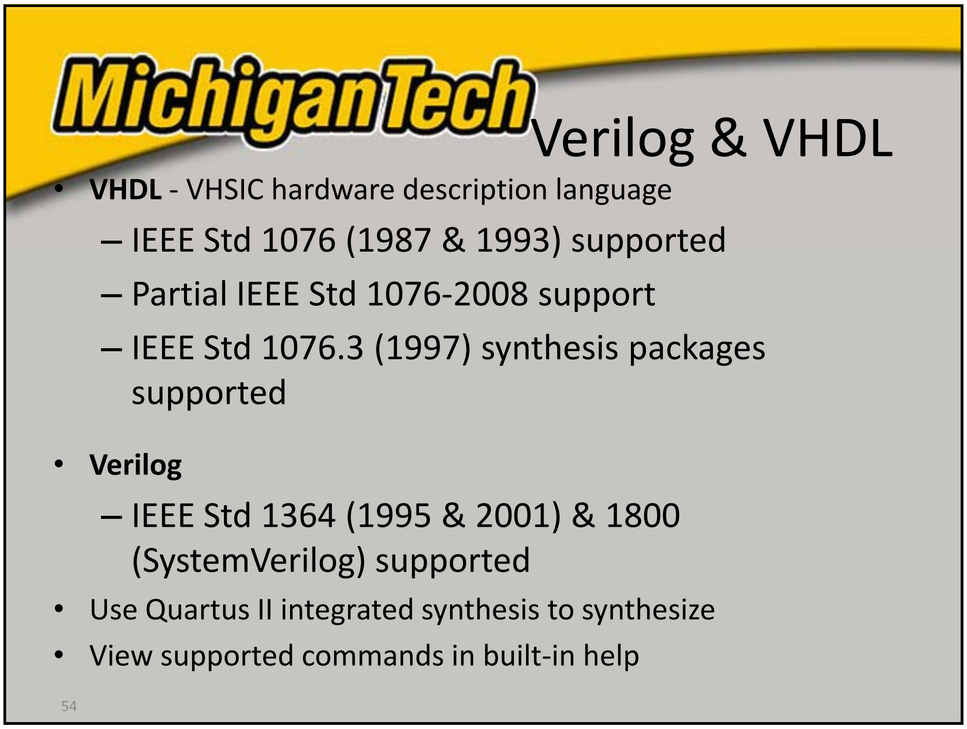
File ⇒ New or 🗋 in Toolbar



Text Design Entry

- Quartus II Text Editor features
  - Block commenting
  - Line numbering in HDL text files
  - Bookmarks
  - Syntax coloring
  - Find/replace text
  - Find and highlight matching delimiters
  - Function collapse/expand
  - Create & edit .sdc files for TimeQuest timing analyzer (described later)
  - Preview/editing of full design and construct HDL templates
- Enter text description
  - VHDL (.vhd, .vhdl)
  - Verilog (.v, .vlg, .Verilog, .vh)
  - SystemVerilog (.sv)

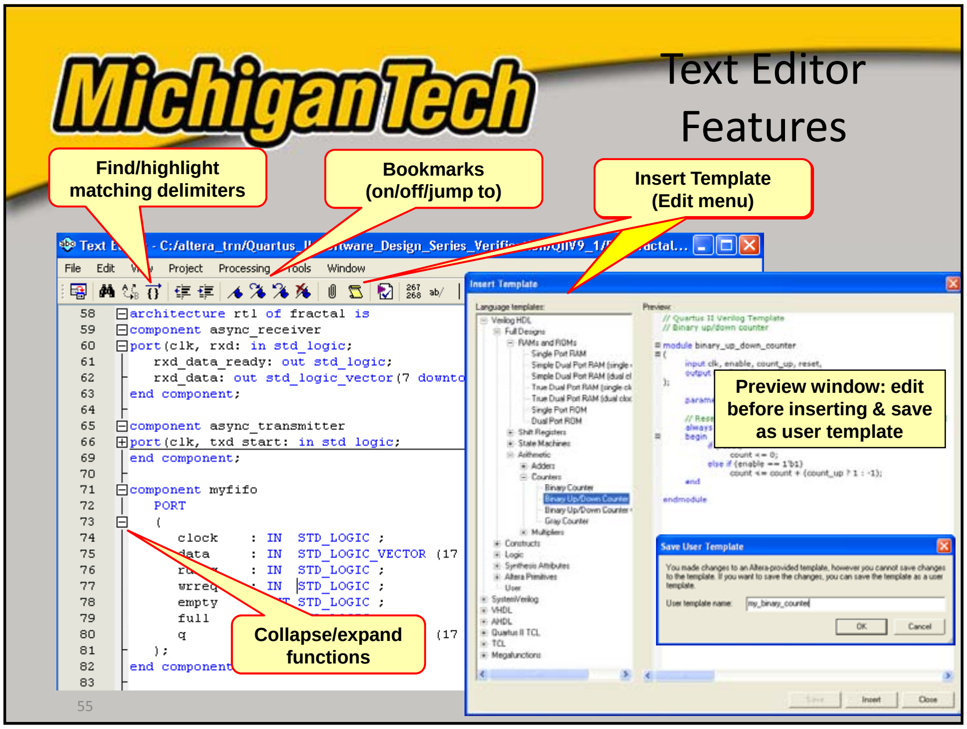
53



Verilog & VHDL

- **VHDL** - VHSIC hardware description language
  - IEEE Std 1076 (1987 & 1993) supported
  - Partial IEEE Std 1076-2008 support
  - IEEE Std 1076.3 (1997) synthesis packages supported
- **Verilog**
  - IEEE Std 1364 (1995 & 2001) & 1800 (SystemVerilog) supported
- Use Quartus II integrated synthesis to synthesize
- View supported commands in built-in help

54



Text Editor Features

Find/highlight matching delimiters

Bookmarks (on/off/jump to)

Insert Template (Edit menu)

Text E... - C:/altera_trn/Quartus_II...tware_Design_Series_Verific.../QIIV9_1/...actal...
File
Edit
View
Project
Processing
Tools
Window
267
268
ab/

```
architecture rtl of fractal is
component async_receiver
port(clk, rxd: in std_logic;
   rxd_data_ready: out std_logic;
   rxd_data: out std_logic_vector(7 downto
end component;

component async_transmitter
port(clk, txd start: in std logic;
end component;

component myfifo
   PORT
   (
      clock    : IN  STD_LOGIC ;
      data     : IN  STD_LOGIC_VECTOR (17
      rdreq    : IN  STD_LOGIC ;
      wrreq    : IN  STD_LOGIC ;
      empty    : OUT STD_LOGIC ;
      full
      q                                (17
   );
end component
```

```
// Quartus II Verilog Template
// Binary up/down counter

module binary_up_down_counter
(
    input clk, enable, count_up, reset,
    output
);

    param

    // Rese
    always
    begin
            count <= 0;
        else if (enable == 1'b1)
            count <= count + (count_up ? 1 : -1);
    end

endmodule
```

Preview window: edit before inserting & save as user template

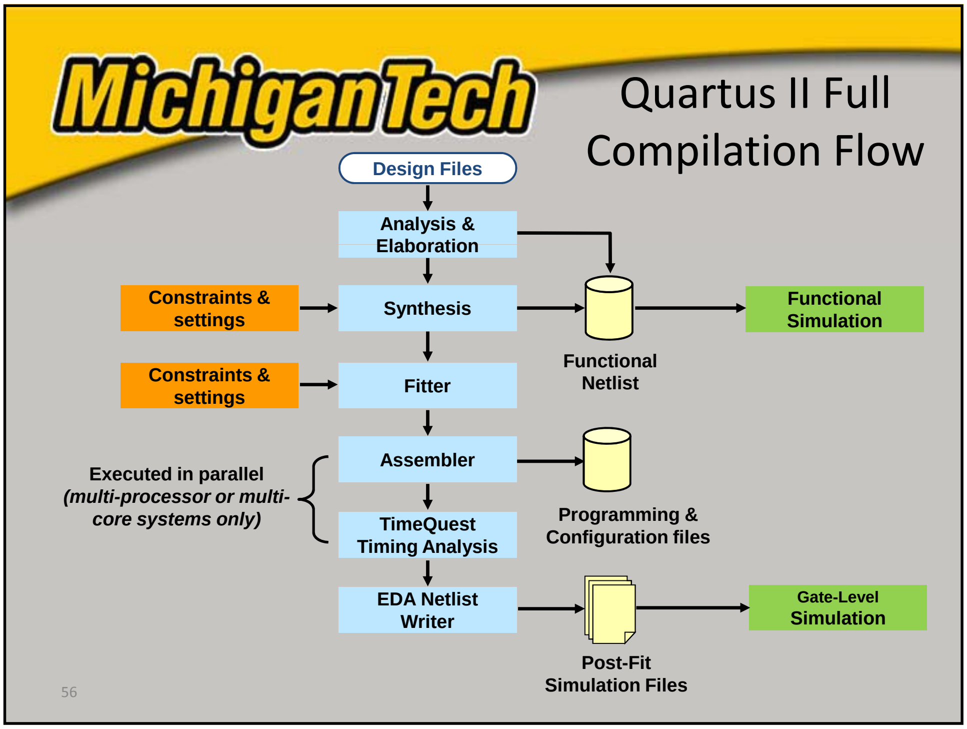
Save User Template

You made changes to an Altera-provided template, however you cannot save changes to the template. If you want to save the changes, you can save the template as a user template.

User template name: my_binary_counter

Collapse/expand functions

55



Quartus II Full Compilation Flow

Design Files

Analysis & Elaboration

Constraints & settings

Synthesis

Functional Netlist

Functional Simulation

Constraints & settings

Fitter

Assembler

Programming & Configuration files

Executed in parallel *(multi-processor or multi-core systems only)*

TimeQuest Timing Analysis

EDA Netlist Writer

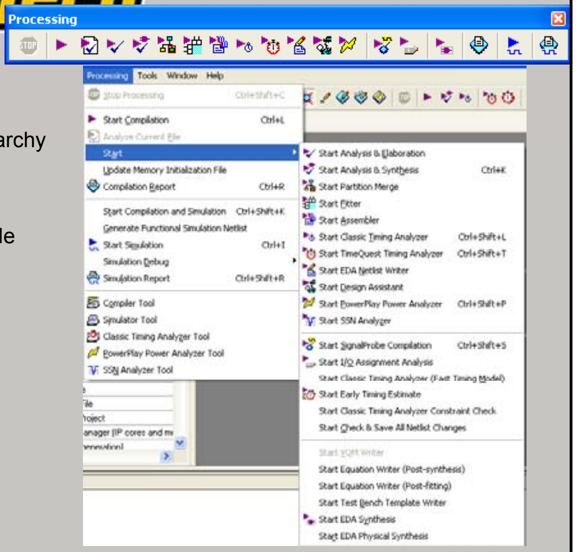
Post-Fit Simulation Files

Gate-Level Simulation

56

Processing Options

- Start Compilation (Full)
- Start Analysis & Elaboration
  - Checks syntax & builds hierarchy
  - Performs initial synthesis
- Start Analysis & Synthesis
  - Synthesizes & optimizes code
- Start Fitter
  - Places & routes design
  - Generates output netlists
- Start Assembler
  - Generate programming files
- Analyzers
  - I/O Assignment
  - PowerPlay
  - SSN (Switching Noise)
- Start Design Assistant

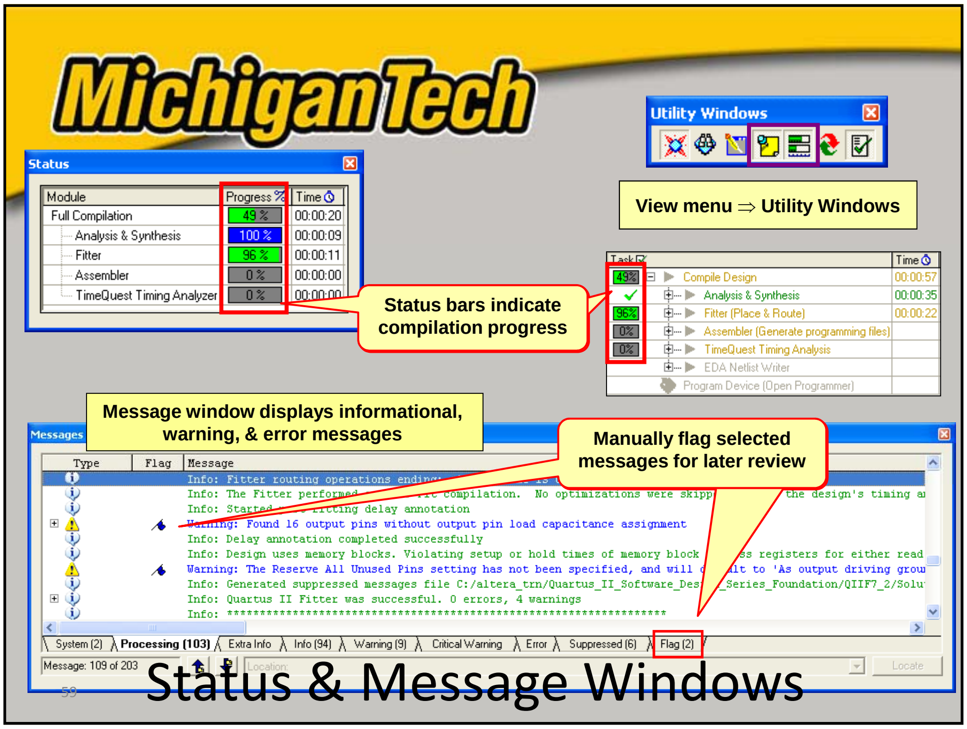

Processing
Processing
Tools
Window
Help
Stop Processing Ctrl+Shift+C
Start Compilation Ctrl+L
Analyze Current File
Start
Update Memory Initialization File
Compilation Report Ctrl+R
Start Compilation and Simulation Ctrl+Shift+K
Generate Functional Simulation Netlist
Start Simulation Ctrl+I
Simulation Debug
Simulation Report Ctrl+Shift+R
Compiler Tool
Simulator Tool
Classic Timing Analyzer Tool
PowerPlay Power Analyzer Tool
SSN Analyzer Tool
Start Analysis & Elaboration
Start Analysis & Synthesis Ctrl+K
Start Partition Merge
Start Fitter
Start Assembler
Start Classic Timing Analyzer Ctrl+Shift+L
Start TimeQuest Timing Analyzer Ctrl+Shift+T
Start EDA Netlist Writer
Start Design Assistant
Start PowerPlay Power Analyzer Ctrl+Shift+P
Start SSN Analyzer
Start SignalProbe Compilation Ctrl+Shift+S
Start I/O Assignment Analysis
Start Classic Timing Analyzer (Fast Timing Model)
Start Early Timing Estimate
Start Classic Timing Analyzer Constraint Check
Start Check & Save All Netlist Changes
Start VQM Writer
Start Equation Writer (Post-synthesis)
Start Equation Writer (Post-fitting)
Start Test Bench Template Writer
Start EDA Synthesis
Start EDA Physical Synthesis

57

Compilation Design Flows

- Default "flat" compilation flow
  - Design compiled as a whole
  - Global optimizations performed
- Incremental flow (on by default for new projects)
  - User assigns design partitions
  - Each partition processed separately; results merged
  - Post-synthesis or post-fit netlists for partitions are reused
  - Benefits
    - Decrease compilation time
    - Preserve compilation results and timing performance
    - Enable faster timing closure

58



Status & Message Windows

Status

Module	Progress %	Time
Full Compilation	49 %	00:00:20
Analysis & Synthesis	100 %	00:00:09
Fitter	96 %	00:00:11
Assembler	0 %	00:00:00
TimeQuest Timing Analyzer	0 %	00:00:00

Utility Windows

View menu ⇒ Utility Windows

Task		Time
49%	Compile Design	00:00:57
✓	Analysis & Synthesis	00:00:35
96%	Fitter (Place & Route)	00:00:22
0%	Assembler (Generate programming files)	
0%	TimeQuest Timing Analysis	
	EDA Netlist Writer	
	Program Device (Open Programmer)	

Status bars indicate compilation progress

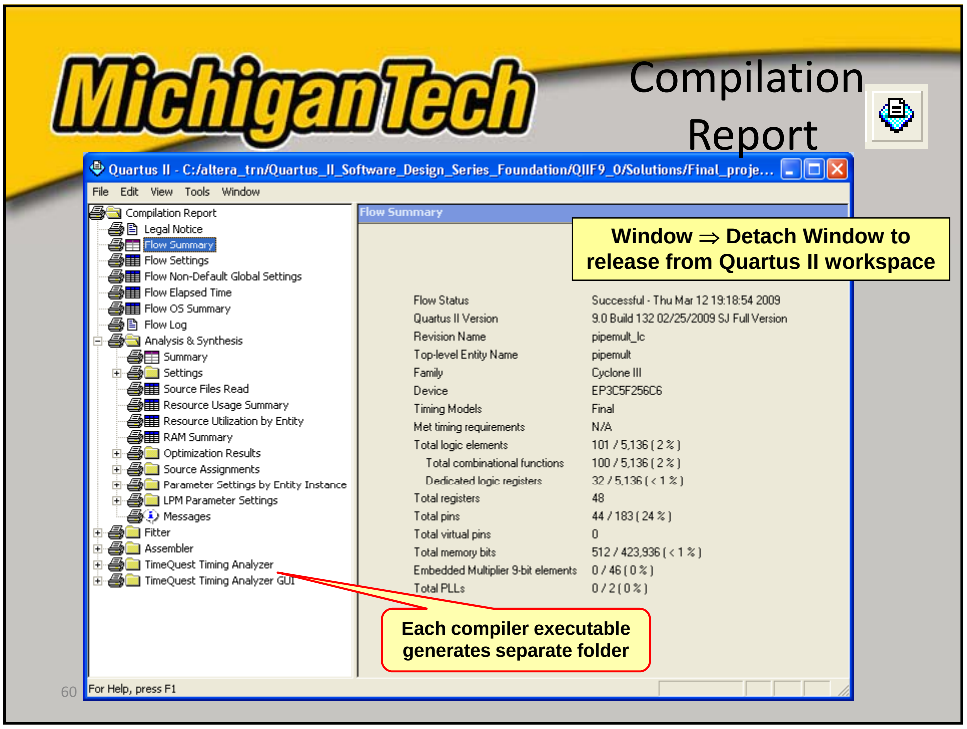
Message window displays informational, warning, & error messages

Manually flag selected messages for later review

Messages

```
Type    Flag   Message
                Info: Fitter routing operations ending
                Info: The Fitter performed          compilation.  No optimizations were skipp          the design's timing a
                Info: Started          fitting delay annotation
                Warning: Found 16 output pins without output pin load capacitance assignment
                Info: Delay annotation completed successfully
                Info: Design uses memory blocks. Violating setup or hold times of memory block          ss registers for either read
                Warning: The Reserve All Unused Pins setting has not been specified, and will d          ult to 'As output driving grou
                Info: Generated suppressed messages file C:/altera_trn/Quartus_II_Software_Des          _Series_Foundation/QIIF7_2/Solu
                Info: Quartus II Fitter was successful. 0 errors, 4 warnings
                Info: ****************************************************************
```

59



Compilation Report

Quartus II - C:/altera_trn/Quartus_II_Software_Design_Series_Foundation/QIIF9_0/Solutions/Final_proje...

- Compilation Report
  - Legal Notice
  - Flow Summary
  - Flow Settings
  - Flow Non-Default Global Settings
  - Flow Elapsed Time
  - Flow OS Summary
  - Flow Log
  - Analysis & Synthesis
    - Summary
    - Settings
    - Source Files Read
    - Resource Usage Summary
    - Resource Utilization by Entity
    - RAM Summary
    - Optimization Results
    - Source Assignments
    - Parameter Settings by Entity Instance
    - LPM Parameter Settings
    - Messages
  - Fitter
  - Assembler
  - TimeQuest Timing Analyzer
  - TimeQuest Timing Analyzer GUI

Flow Summary

Flow Status	Successful - Thu Mar 12 19:18:54 2009
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	pipemult_lc
Top-level Entity Name	pipemult
Family	Cyclone III
Device	EP3C5F256C6
Timing Models	Final
Met timing requirements	N/A
Total logic elements	101 / 5,136 (2 %)
Total combinational functions	100 / 5,136 (2 %)
Dedicated logic registers	32 / 5,136 (< 1 %)
Total registers	48
Total pins	44 / 183 (24 %)
Total virtual pins	0
Total memory bits	512 / 423,936 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 46 (0 %)
Total PLLs	0 / 2 (0 %)

Window ⇒ Detach Window to release from Quartus II workspace

Each compiler executable generates separate folder

For Help, press F1

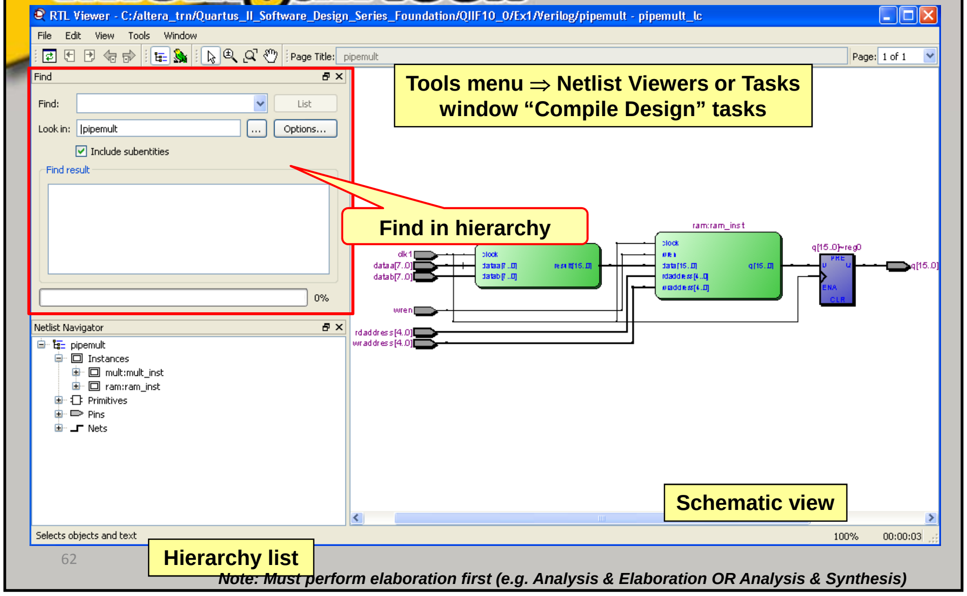
60

Netlist Viewers

- RTL Viewer
  - Schematic of design after Analysis and Elaboration
  - Visually check initial HDL before synthesis optimizations
  - Locate synthesized nodes for assigning constraints
  - Debug verification issues
- Technology Map Viewers (Post-Mapping or Post-Fitting)
  - Graphically represents results of mapping (post-synthesis) & fitting
  - Analyze critical timing paths graphically
  - Locate nodes & node names after optimizations (cross-probing)

61

RTL Viewer



Page Title: pipemult

Page: 1 of 1

Find:

Look in: |pipemult

☑ Include subentities

0%

Selects objects and text

100%

00:00:03

Tools menu ⇒ Netlist Viewers or Tasks window “Compile Design” tasks

Find in hierarchy

Schematic view

Hierarchy list

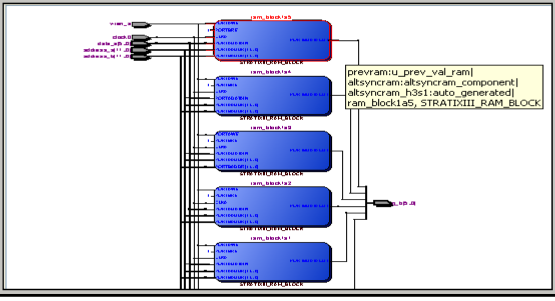
62

Note: Must perform elaboration first (e.g. Analysis & Elaboration OR Analysis & Synthesis)

MichiganTech

Schematic View (RTL Viewer)

- Represents design using logic blocks & nets
  - I/O pins
  - Registers
  - Muxes
  - Gates (AND, OR, etc.)
  - Operators (adders, multipliers, etc.)



prevram:u_prev_val_ram|
altsyncram:altsyncram_component|
altsyncram_h3s1:auto_generated|
ram_block1a5, STRATIXIII_RAM_BLOCK

Place pointer over any element in schematic to see details
- **Name**
- **Internal resource count**

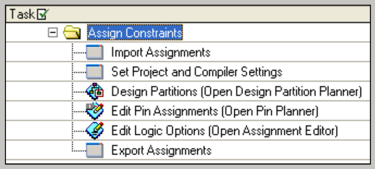
MichiganTech

Compilation Summary

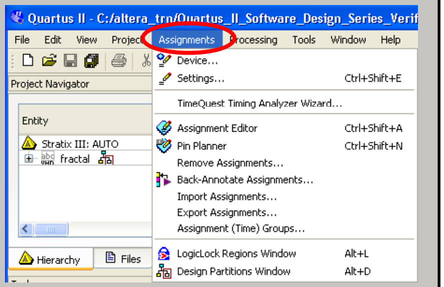
- Compilation includes synthesis & fitting
- Compilation Report contains detailed information on compilation results
- Use Quartus II software tools to understand how design was processed
  - RTL Viewer
  - Technology Map Viewers
  - State Machine Viewer
  - Chip Planner
  - Resource Property Editors

Synthesis & Fitting Control

- Controlled using two methods
  - Settings - Project-wide switches
  - Assignments - Individual entity/node controls
- Both accessed in **Assignments** menu or Tasks window
- Stored in QSF file for project/revision

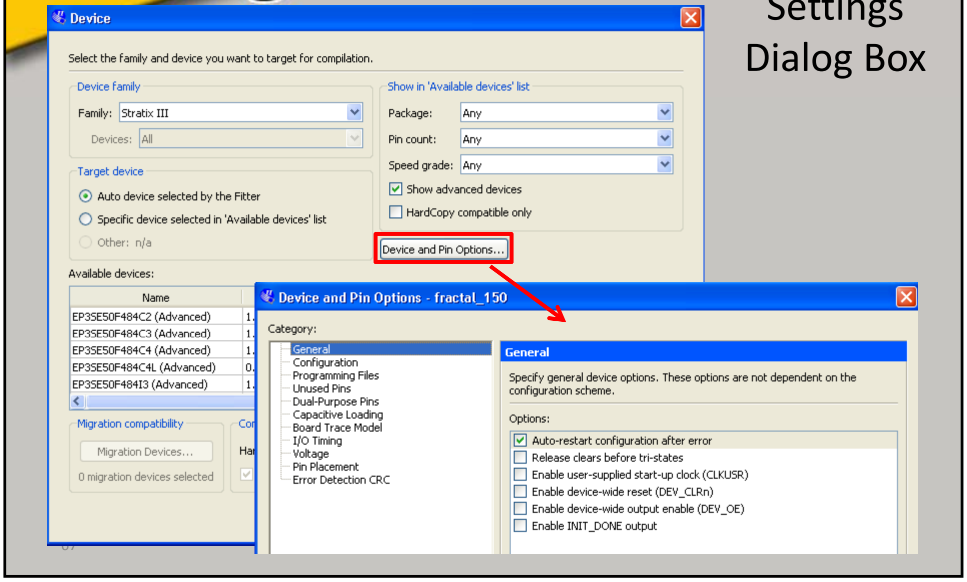


Task
- Assign Constraints
  - Import Assignments
  - Set Project and Compiler Settings
  - Design Partitions (Open Design Partition Planner)
  - Edit Pin Assignments (Open Pin Planner)
  - Edit Logic Options (Open Assignment Editor)
  - Export Assignments


Quartus II - C:/altera_trn/Quartus_II_Software_Design_Series_Verif
File
Edit
View
Project
Assignments
Processing
Tools
Window
Help
Project Navigator
Entity
Stratix III: AUTO
fractal
Hierarchy
Files
Device...
Settings... Ctrl+Shift+E
TimeQuest Timing Analyzer Wizard...
Assignment Editor Ctrl+Shift+A
Pin Planner Ctrl+Shift+N
Remove Assignments...
Back-Annotate Assignments...
Import Assignments...
Export Assignments...
Assignment (Time) Groups...
LogicLock Regions Window Alt+L
Design Partitions Window Alt+D

Settings

- Project-wide switches that affect entire design
- Examples
  - Device selection
  - Synthesis optimization
  - Fitter settings
  - Physical synthesis
  - Design Assistant
- Located in Settings dialog box
  - **Assignments** menu
  - **Set Project and Compiler Settings** task in Tasks window



Device

Select the family and device you want to target for compilation.

Device family

Family: Stratix III

Devices: All

Target device

● Auto device selected by the Fitter

○ Specific device selected in 'Available devices' list

○ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

☑ Show advanced devices

□ HardCopy compatible only

Device and Pin Options...

Available devices:

Name	
EP3SE50F484C2 (Advanced)	1.
EP3SE50F484C3 (Advanced)	1.
EP3SE50F484C4 (Advanced)	1.
EP3SE50F484C4L (Advanced)	0.
EP3SE50F484I3 (Advanced)	1.

Migration compatibility

Migration Devices...

0 migration devices selected

Device and Pin Options - fractal_150

Category:

General
Configuration
Programming Files
Unused Pins
Dual-Purpose Pins
Capacitive Loading
Board Trace Model
I/O Timing
Voltage
Pin Placement
Error Detection CRC

General

Specify general device options. These options are not dependent on the configuration scheme.

Options:

☑ Auto-restart configuration after error

□ Release clears before tri-states

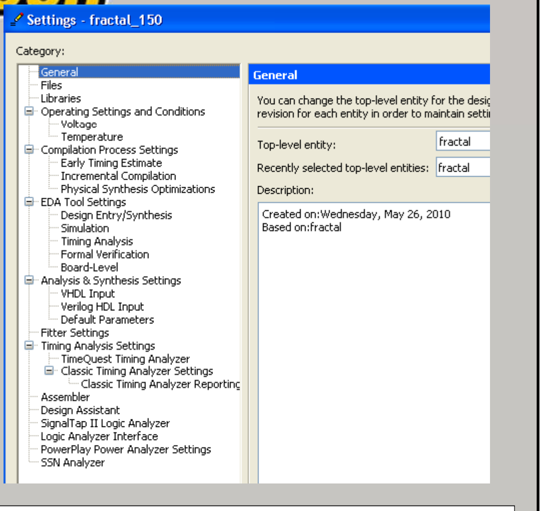
□ Enable user-supplied start-up clock (CLKUSR)

□ Enable device-wide reset (DEV_CLRn)

□ Enable device-wide output enable (DEV_OE)

□ Enable INIT_DONE output

Device Settings Dialog Box



Settings - fractal_150

Category:

General
Files
Libraries
Operating Settings and Conditions
  Voltage
  Temperature
Compilation Process Settings
  Early Timing Estimate
  Incremental Compilation
  Physical Synthesis Optimizations
EDA Tool Settings
  Design Entry/Synthesis
  Simulation
  Timing Analysis
  Formal Verification
  Board-Level
Analysis & Synthesis Settings
  VHDL Input
  Verilog HDL Input
  Default Parameters
Fitter Settings
Timing Analysis Settings
  TimeQuest Timing Analyzer
  Classic Timing Analyzer Settings
    Classic Timing Analyzer Reporting
Assembler
Design Assistant
SignalTap II Logic Analyzer
Logic Analyzer Interface
PowerPlay Power Analyzer Settings
SSN Analyzer

General

You can change the top-level entity for the desig
revision for each entity in order to maintain setti

Top-level entity: fractal

Recently selected top-level entities: fractal

Description:

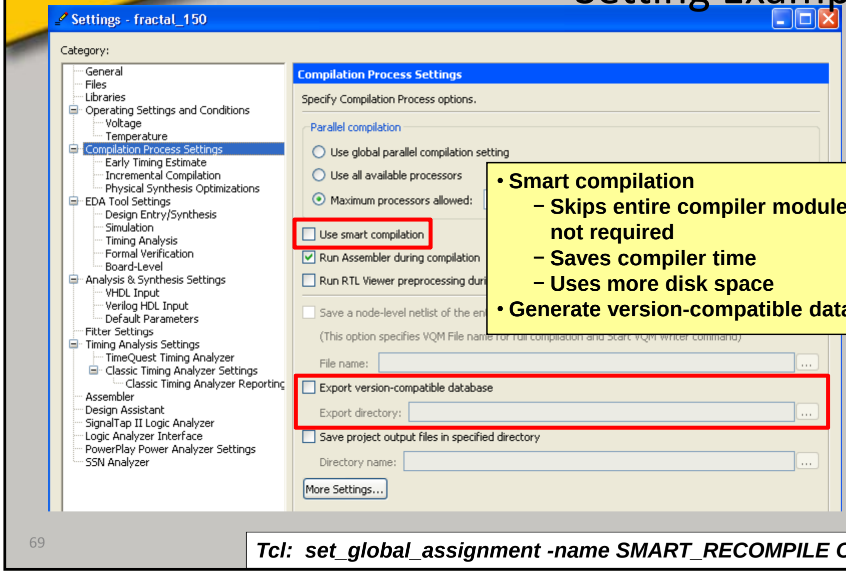
Created on:Wednesday, May 26, 2010
Based on:fractal

<u>Change settings</u>
- **Top-level entity**
- **Add/remove files**
- **Libraries**
- **Compiler settings**
- **EDA tool settings**
- **Synthesis settings**
- **Fitter settings**
- **Timing Analysis settings**
- **Power analysis settings**
- **SSN Analyzer settings**

Settings Dialog Box

Tcl: set_global_assignment –name <assignment_name*> <value>

Compilation Process Setting Examples



Settings - fractal_150

Category:

- General
- Files
- Libraries
- Operating Settings and Conditions
  - Voltage
  - Temperature
- Compilation Process Settings
  - Early Timing Estimate
  - Incremental Compilation
  - Physical Synthesis Optimizations
- EDA Tool Settings
  - Design Entry/Synthesis
  - Simulation
  - Timing Analysis
  - Formal Verification
  - Board-Level
- Analysis & Synthesis Settings
  - VHDL Input
  - Verilog HDL Input
  - Default Parameters
- Fitter Settings
- Timing Analysis Settings
  - TimeQuest Timing Analyzer
  - Classic Timing Analyzer Settings
    - Classic Timing Analyzer Reporting
- Assembler
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- PowerPlay Power Analyzer Settings
- SSN Analyzer

Compilation Process Settings

Specify Compilation Process options.

Parallel compilation

○ Use global parallel compilation setting

○ Use all available processors

● Maximum processors allowed:

□ Use smart compilation

☑ Run Assembler during compilation

□ Run RTL Viewer preprocessing duri

□ Save a node-level netlist of the en

(This option specifies VQM File name for full compilation and Start VQM Writer command)

File name:

□ Export version-compatible database

Export directory:

□ Save project output files in specified directory

Directory name:

More Settings...

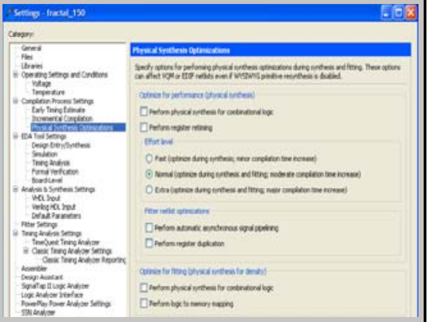
- **Smart compilation**
  - **Skips entire compiler modules when not required**
  - **Saves compiler time**
  - **Uses more disk space**
- **Generate version-compatible database**

69

Tcl: set_global_assignment -name SMART_RECOMPILE ON

Physical Synthesis

- Optimize during synthesis or re-synthesize based on Fitter output
  - Makes incremental changes that improve results for a given placement
  - Compensates for routing delays from Fitter
  - Apply globally (Settings) or only to specific design entities (Assignment Editor)



Settings - fractal_150

Category:

- General
- Files
- Libraries
- Operating Settings and Conditions
  - Voltage
  - Temperature
- Compilation Process Settings
  - Early Timing Estimate
  - Incremental Compilation
  - Physical Synthesis Optimizations
- EDA Tool Settings
  - Design Entry/Synthesis
  - Simulation
  - Timing Analysis
  - Formal Verification
  - Board-Level
- Analysis & Synthesis Settings
  - VHDL Input
  - Verilog HDL Input
  - Default Parameters
- Fitter Settings
- Timing Analysis Settings
  - TimeQuest Timing Analyzer
  - Classic Timing Analyzer Settings
    - Classic Timing Analyzer Reporting
- Assembler
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- PowerPlay Power Analyzer Settings
- SSN Analyzer

Physical Synthesis Optimizations

Specify options for performing physical synthesis optimizations during synthesis and fitting. These options can affect VQM or EDIF netlists even if WYSIWYG primitive resynthesis is disabled.

Optimize for performance (physical synthesis)

□ Perform physical synthesis for combinational logic

□ Perform register retiming

Effort level

○ Fast (optimize during synthesis; minor compilation time increase)

● Normal (optimize during synthesis and fitting; moderate compilation time increase)

○ Extra (optimize during synthesis and fitting; major compilation time increase)

Fitter netlist optimizations

□ Perform automatic asynchronous signal pipelining

□ Perform register duplication

Optimize for fitting (physical synthesis for density)

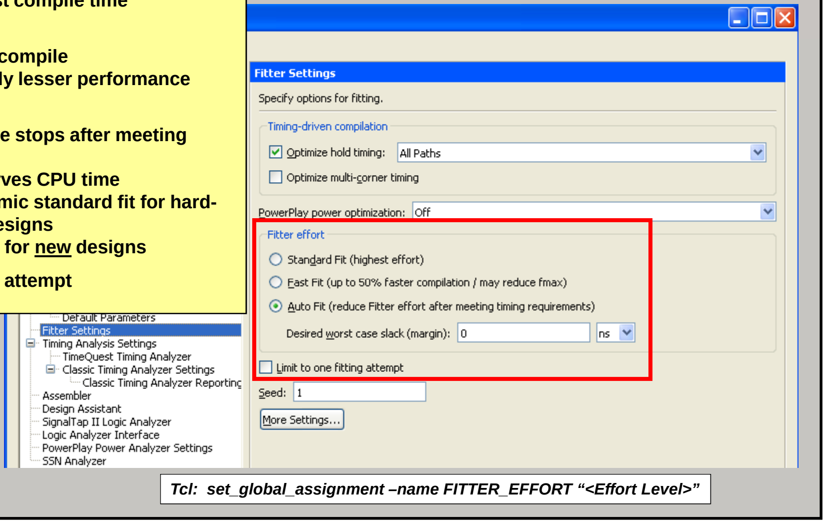
□ Perform physical synthesis for combinational logic

□ Perform logic to memory mapping

70

Fitter Settings – Fitter Effort

- Standard Fit
  - Highest effort
  - Longest compile time
- Fast Fit
  - Faster compile
  - Possibly lesser performance
- Auto Fit
  - Compile stops after meeting timing
  - Conserves CPU time
  - Will mimic standard fit for hard-to-fit designs
  - Default for <u>new</u> designs
- One fitting attempt


Default Parameters
Fitter Settings
Timing Analysis Settings
TimeQuest Timing Analyzer
Classic Timing Analyzer Settings
Classic Timing Analyzer Reporting
Assembler
Design Assistant
SignalTap II Logic Analyzer
Logic Analyzer Interface
PowerPlay Power Analyzer Settings
SSN Analyzer

Fitter Settings

Specify options for fitting.

Timing-driven compilation

☑ Optimize hold timing: All Paths

□ Optimize multi-corner timing

PowerPlay power optimization: Off

Fitter effort

○ Standard Fit (highest effort)

○ Fast Fit (up to 50% faster compilation / may reduce fmax)

● Auto Fit (reduce Fitter effort after meeting timing requirements)

Desired worst case slack (margin): 0 ns

□ Limit to one fitting attempt

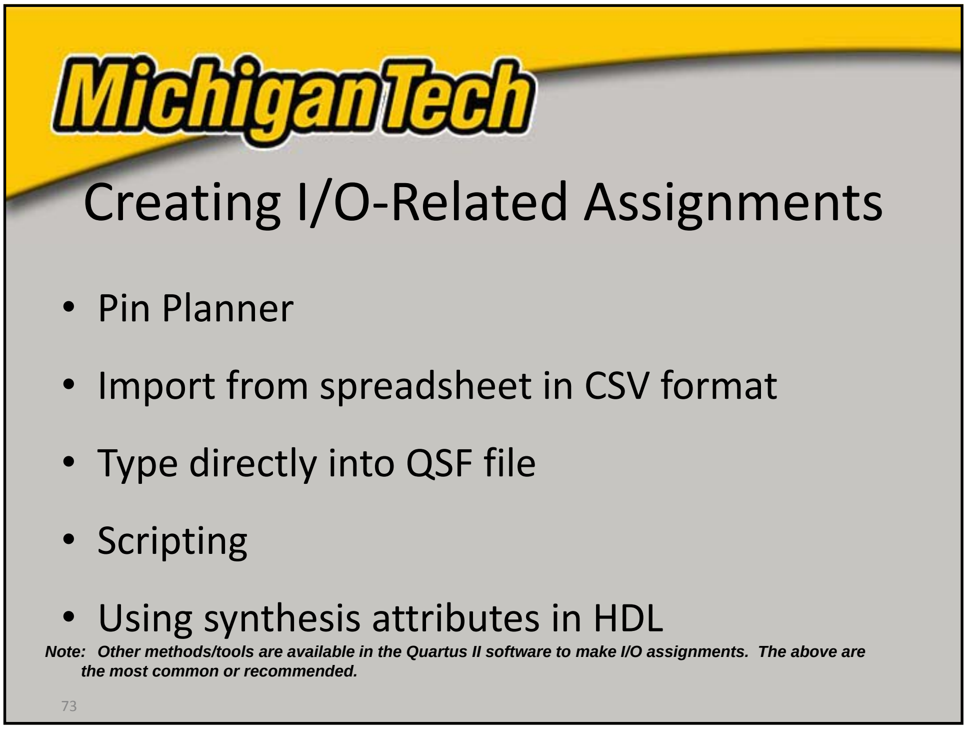
Seed: 1

More Settings...

Tcl: set_global_assignment –name FITTER_EFFORT "<Effort Level>"

I/O Planning Need

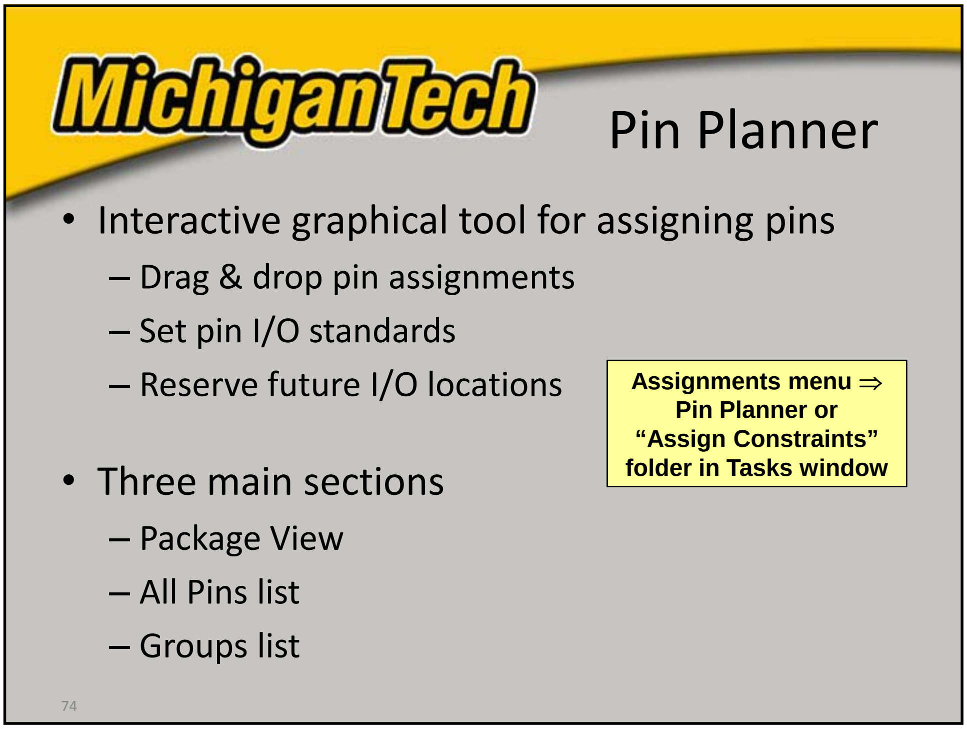
- I/O standards increasing in complexity
- FPGA/CPLD I/O structure increasing in complexity
  - Results in increased pin placement guidelines
- PCB development performed simultaneously with FPGA design
- Pin assignments need to be verified earlier in design cycle
- Designers need easy way to transfer pin assignments into board tools



Creating I/O-Related Assignments

- Pin Planner
- Import from spreadsheet in CSV format
- Type directly into QSF file
- Scripting
- Using synthesis attributes in HDL

***Note:** Other methods/tools are available in the Quartus II software to make I/O assignments. The above are the most common or recommended.*

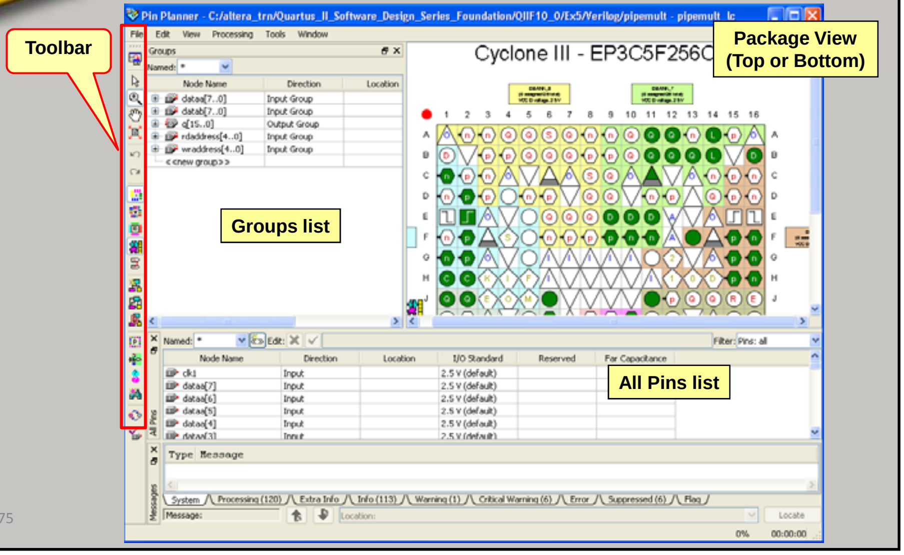


Pin Planner

- Interactive graphical tool for assigning pins
  - Drag & drop pin assignments
  - Set pin I/O standards
  - Reserve future I/O locations
- Three main sections
  - Package View
  - All Pins list
  - Groups list

Assignments menu ⇒ Pin Planner or "Assign Constraints" folder in Tasks window

Pin Planner Window (1)



Pin Planner - C:/altera_trn/Quartus_II_Software_Design_Series_Foundation/QIIF10_0/Ex5/Verilog/pipemult - pipemult_lc

Toolbar

Package View (Top or Bottom)

Groups list

All Pins list

Named: *

Node Name	Direction	Location
dataa[7..0]	Input Group	
datab[7..0]	Input Group	
q[15..0]	Output Group	
rdaddress[4..0]	Input Group	
wraddress[4..0]	Input Group	
<< new group >>		

Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Standard	Reserved	Far Capacitance
clk1	Input		2.5 V (default)		
dataa[7]	Input		2.5 V (default)		
dataa[6]	Input		2.5 V (default)		
dataa[5]	Input		2.5 V (default)		
dataa[4]	Input		2.5 V (default)		
dataa[3]	Input		2.5 V (default)		

Type Message

System / Processing (120) / Extra Info / Info (113) / Warning (1) / Critical Warning (6) / Error / Suppressed (6) / Flag

Message: Location: Locate

0% 00:00:00

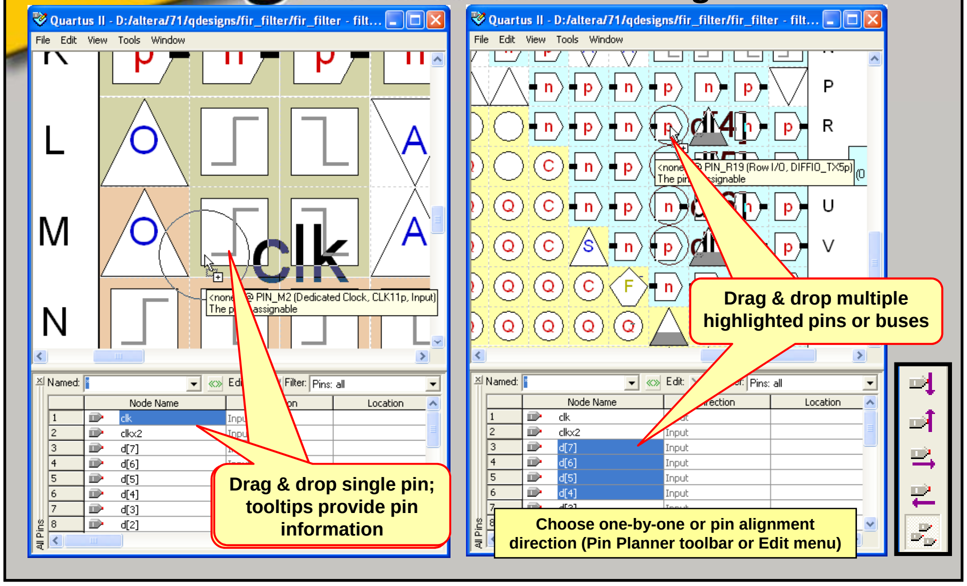
75

Pin Planner Window (2)

- Package View
  - Displays graphical representation of chip package
  - Use to make or edit design I/O assignments
  - Use to locate other package pins (power & configuration pins)
- All Pins list
  - Displays I/O pins (signals) in design as indicated by filter
  - Use to edit pin settings/properties directly
- Groups list
  - Similar to All Pins list except displays only groups & buses
  - Use to make bus and group assignments
  - Use to create new user-defined groups

76

Assigning Pin Locations Using Pin Planner



Quartus II - D:/altera/71/qdesigns/fir_filter/fir_filter - filt...

<none> @ PIN_M2 (Dedicated Clock, CLK11p, Input)
The p[...] assignable

<none> @ PIN_R19 (Row I/O, DIFFIO_TX5p)
The pin[...]signable

Named: * Edi[t] Filter: Pins: all

	Node Name	[Directi]on	Location
1	clk	Inpu[t]	
2	clkx2	Inpu[t]	
3	d[7]		
4	d[6]		
5	d[5]		
6	d[4]		
7	d[3]		
8	d[2]		

Named: * Edit: [Filt]er: Pins: all

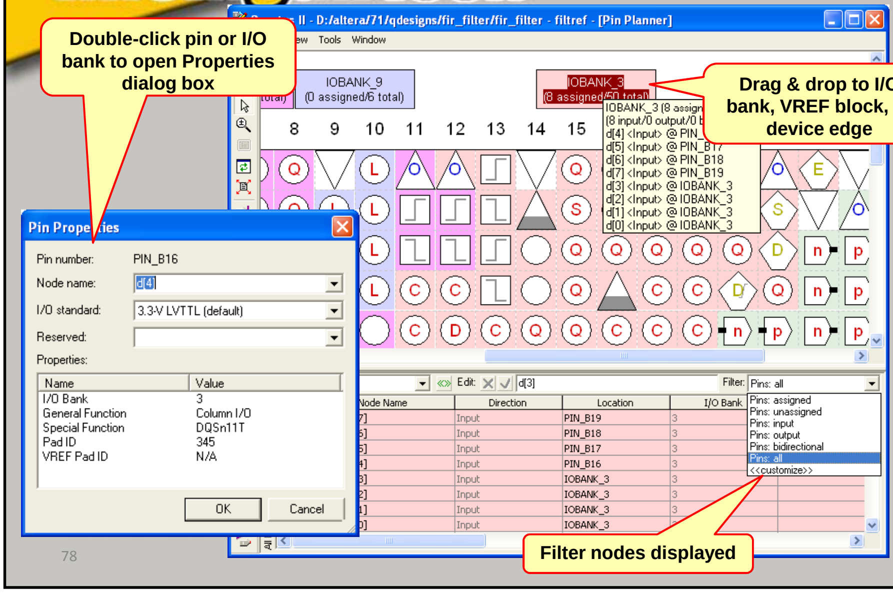
	Node Name	[Di]rection	Location
1	clk	[Inp]ut	
2	clkx2	Input	
3	d[7]	Input	
4	d[6]	Input	
5	d[5]	Input	
6	d[4]	Input	
7	d[3]	Input	

Drag & drop single pin; tooltips provide pin information

Drag & drop multiple highlighted pins or buses

Choose one-by-one or pin alignment direction (Pin Planner toolbar or Edit menu)

Assigning Pin Locations Using Pin Planner (2)



[Quartus] II - D:/altera/71/qdesigns/fir_filter/fir_filter - filtref - [Pin Planner]

IOBANK_3 (8 assign[ed...]
(8 input/0 output/0 b[...]
d[4] <Input> @ PIN_[B16]
d[5] <Input> @ PIN_B17
d[6] <Input> @ PIN_B18
d[7] <Input> @ PIN_B19
d[3] <Input> @ IOBANK_3
d[2] <Input> @ IOBANK_3
d[1] <Input> @ IOBANK_3
d[0] <Input> @ IOBANK_3

Pin Properties

Pin number: PIN_B16

Node name: d[4]

I/O standard: 3.3-V LVTTL (default)

Reserved:

Properties:

Name	Value
I/O Bank	3
General Function	Column I/O
Special Function	DQSn11T
Pad ID	345
VREF Pad ID	N/A

Edit: d[3] Filter: Pins: all

Node Name	Direction	Location	I/O Bank
[d[]7]	Input	PIN_B19	3
[d[]6]	Input	PIN_B18	3
[d[]5]	Input	PIN_B17	3
[d[]4]	Input	PIN_B16	3
[d[]3]	Input	IOBANK_3	3
[d[]2]	Input	IOBANK_3	3
[d[]1]	Input	IOBANK_3	3
[d[]0]	Input	IOBANK_3	

Double-click pin or I/O bank to open Properties dialog box

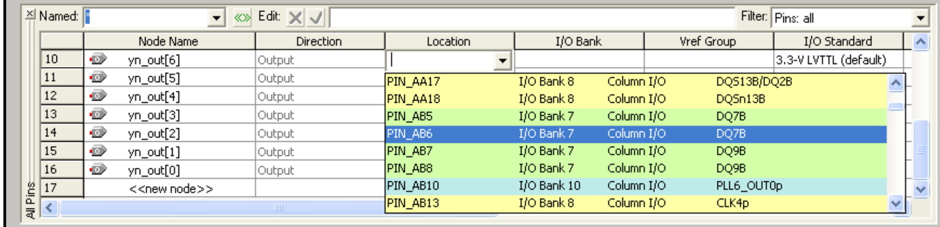
Drag & drop to I/O bank, VREF block, or device edge

Filter nodes displayed

78

Assigning Pin Locations Using Pin Planner (3)

- Select available locations from list of pins color-coded by I/O bank



Named: *   Edit: Filter: Pins: all

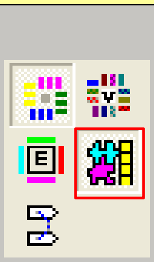
	Node Name	Direction	Location	I/O Bank	Vref Group	I/O Standard
10	yn_out[6]	Output				3.3-V LVTTL (default)
11	yn_out[5]	Output				
12	yn_out[4]	Output				
13	yn_out[3]	Output				
14	yn_out[2]	Output				
15	yn_out[1]	Output				
16	yn_out[0]	Output				
17	<<new node>>					

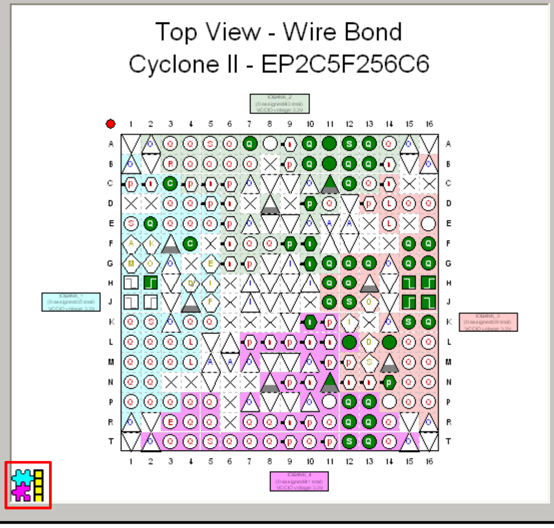
PIN_AA17	I/O Bank 8	Column I/O	DQS13B/DQ2B
PIN_AA18	I/O Bank 8	Column I/O	DQSn13B
PIN_AB5	I/O Bank 7	Column I/O	DQ7B
PIN_AB6	I/O Bank 7	Column I/O	DQ7B
PIN_AB7	I/O Bank 7	Column I/O	DQ9B
PIN_AB8	I/O Bank 7	Column I/O	DQ9B
PIN_AB10	I/O Bank 10	Column I/O	PLL6_OUT0p
PIN_AB13	I/O Bank 8	Column I/O	CLK4p

Show Fitter Placements

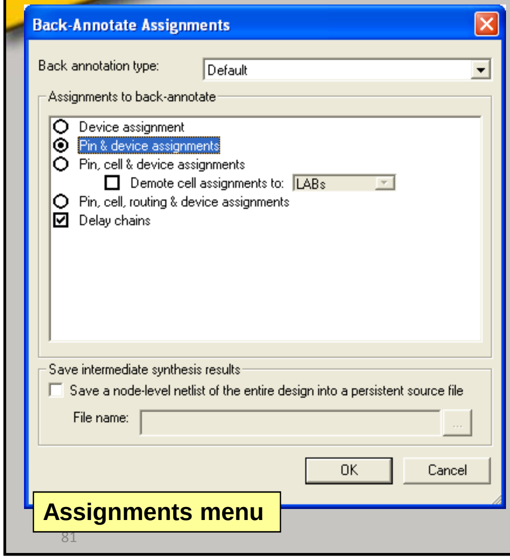
- View I/O locations automatically selected by Fitter

View ⇒ Show
or in Toolbar





Back-Annotation



Back-Annotate Assignments

Back annotation type: Default

Assignments to back-annotate

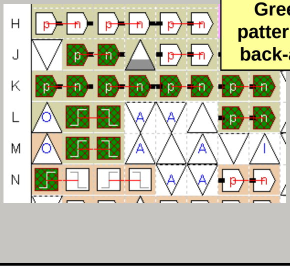
○ Device assignment
● Pin & device assignments
○ Pin, cell & device assignments
□ Demote cell assignments to: LABs
○ Pin, cell, routing & device assignments
☑ Delay chains

Save intermediate synthesis results

□ Save a node-level netlist of the entire design into a persistent source file

File name:

Assignments menu



Green/brown pattern indicates back-annotation

Verifying I/O Assignments

- I/O Assignment Analysis
  - Checks legality of all I/O assignments without full compilation
- Minimal requirements for running
  - I/O declaration
    - HDL port declaration
    - Reserved pin
  - Pin-related assignments
    - I/O standard
    - Current strength
    - Pin location (pin, bank, edge)
    - PCI clamping diode
    - Toggle rate

Processing menu ⇒ Start ⇒ Start I/O Assignment Analysis or Tasks window

Run from Pin Planner toolbar

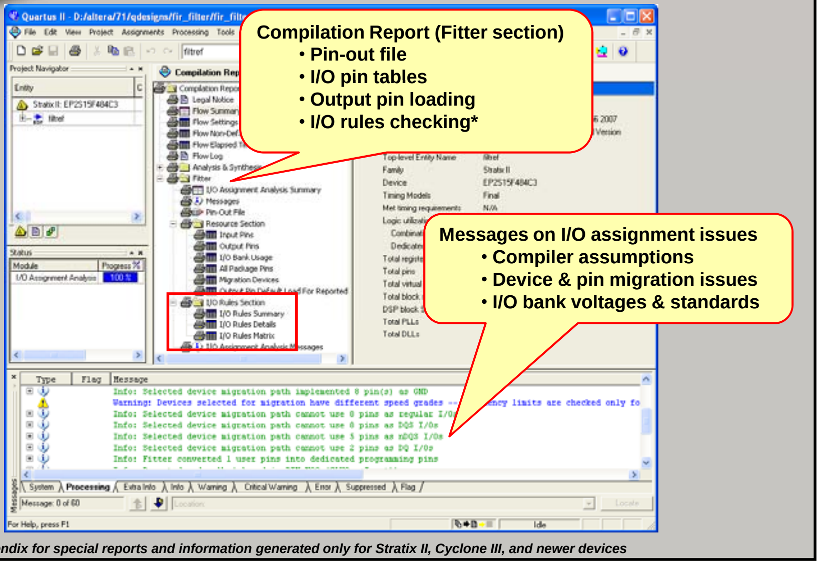
I/O Rules Checked

- No internal logic
  - Checks I/O locations & constraints with respect to other I/O & I/O banks
  - Each I/O bank supports a single V_{CCIO}
- I/O connected to logic
  - Checks I/O locations & constraints with respect to other I/O, I/O banks, & internal resources
  - PLL must be driven by a dedicated clock input pin

Note: When working with design files, synthesize design before running I/O Assignment Analysis

83

I/O Assignment Analysis Output



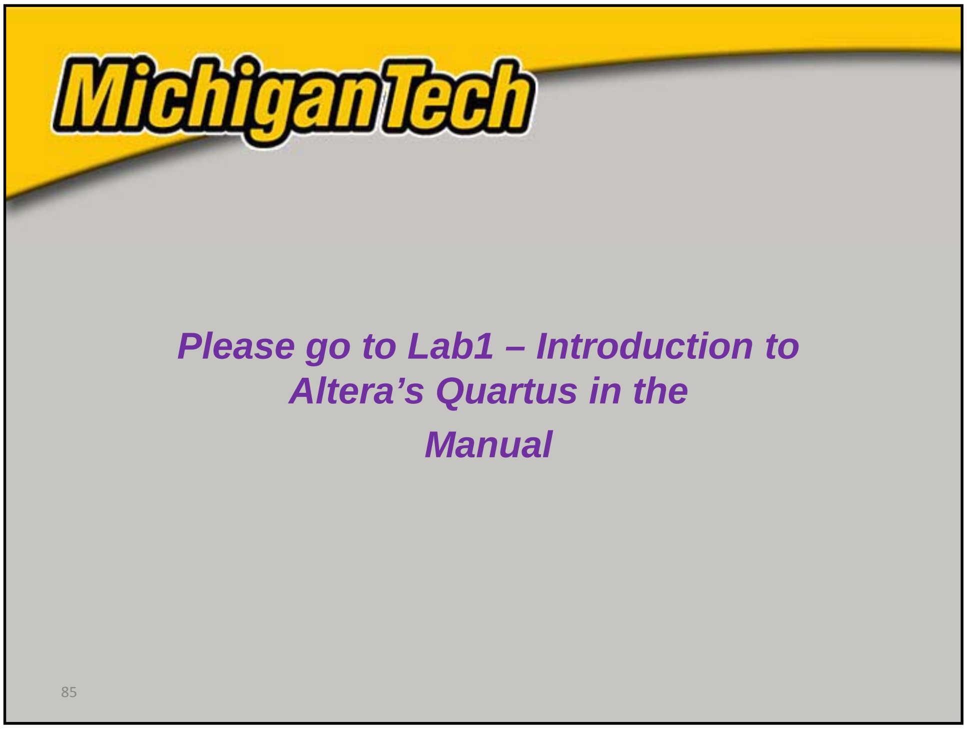
```
Type  Flag  Message
Info: Selected device migration path implemented 8 pin(s) as GND
Warning: Devices selected for migration have different speed grades --      ency limits are checked only fo
Info: Selected device migration path cannot use 0 pins as regular I/O
Info: Selected device migration path cannot use 0 pins as DQS I/Os
Info: Selected device migration path cannot use 5 pins as nDQS I/Os
Info: Selected device migration path cannot use 2 pins as DQ I/Os
Info: Fitter converted 1 user pins into dedicated programming pins
```

Compilation Report (Fitter section)
- **Pin-out file**
- **I/O pin tables**
- **Output pin loading**
- **I/O rules checking***

Messages on I/O assignment issues
- **Compiler assumptions**
- **Device & pin migration issues**
- **I/O bank voltages & standards**

84

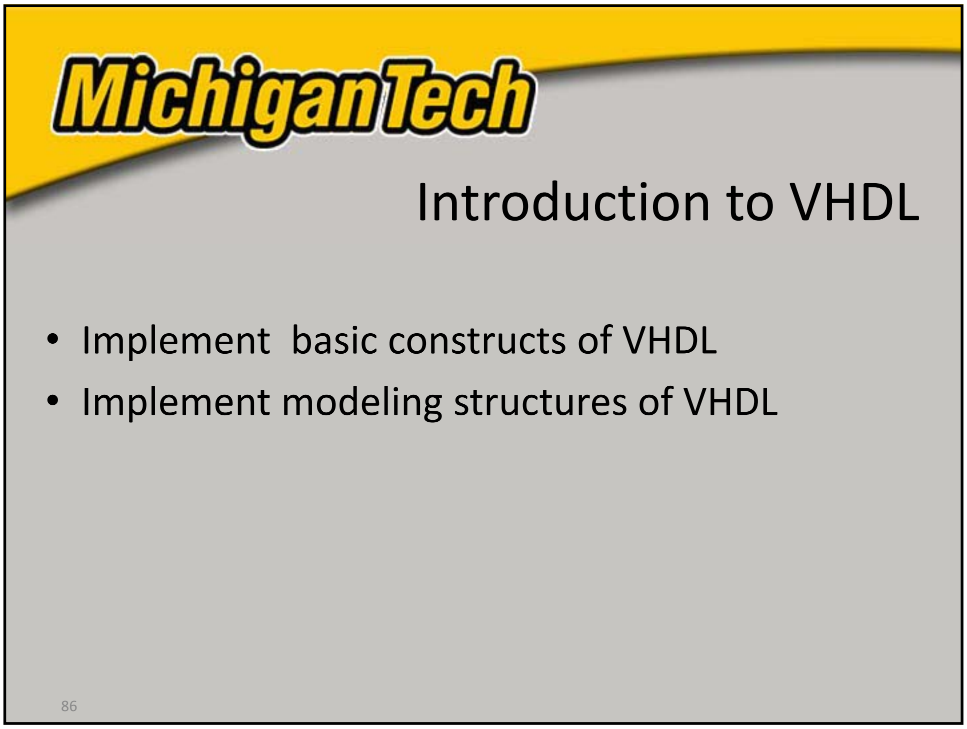
*Note: See Appendix for special reports and information generated only for Stratix II, Cyclone III, and newer devices



Please go to Lab1 – Introduction to Altera's Quartus in the

Manual

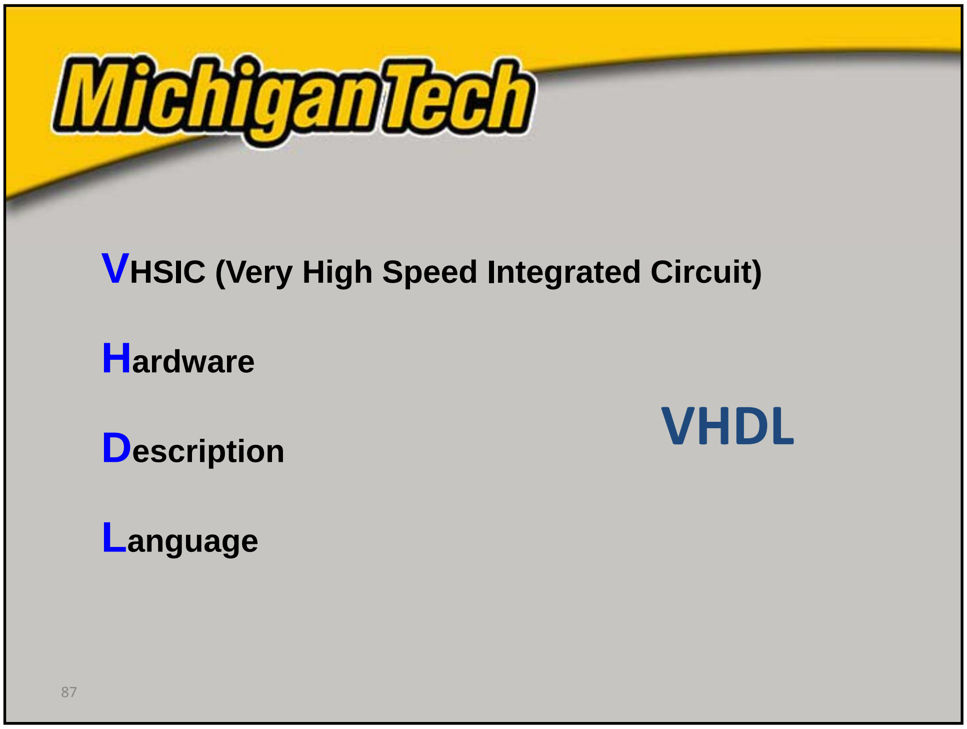
85



Introduction to VHDL

- Implement basic constructs of VHDL
- Implement modeling structures of VHDL

86



VHSIC (Very High Speed Integrated Circuit)

Hardware

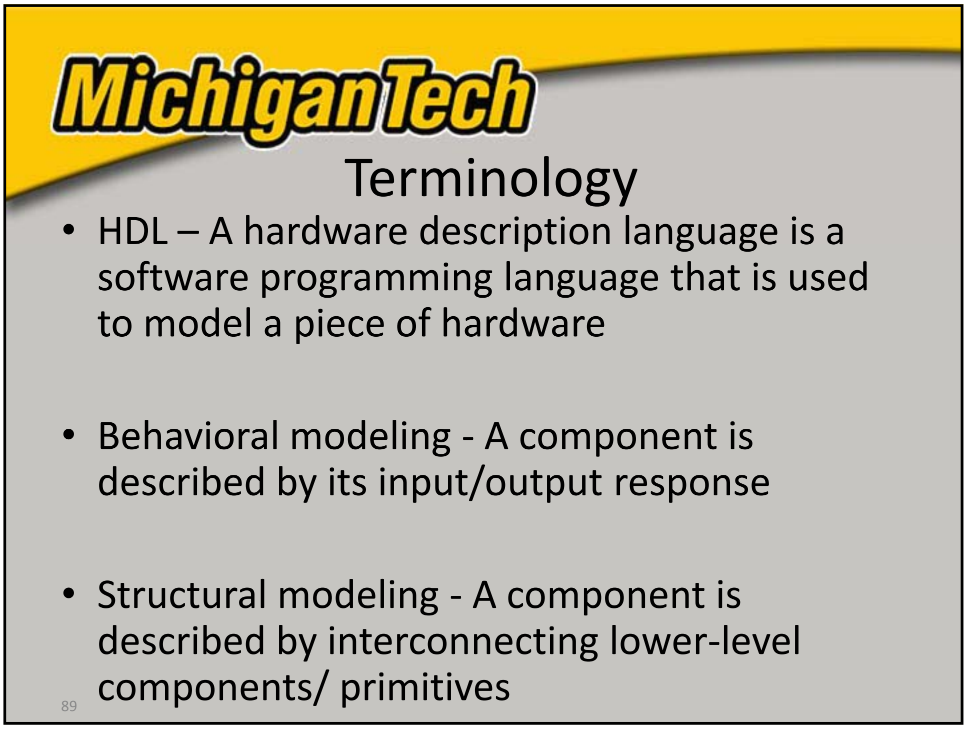
Description

Language

VHDL

What is VHDL?

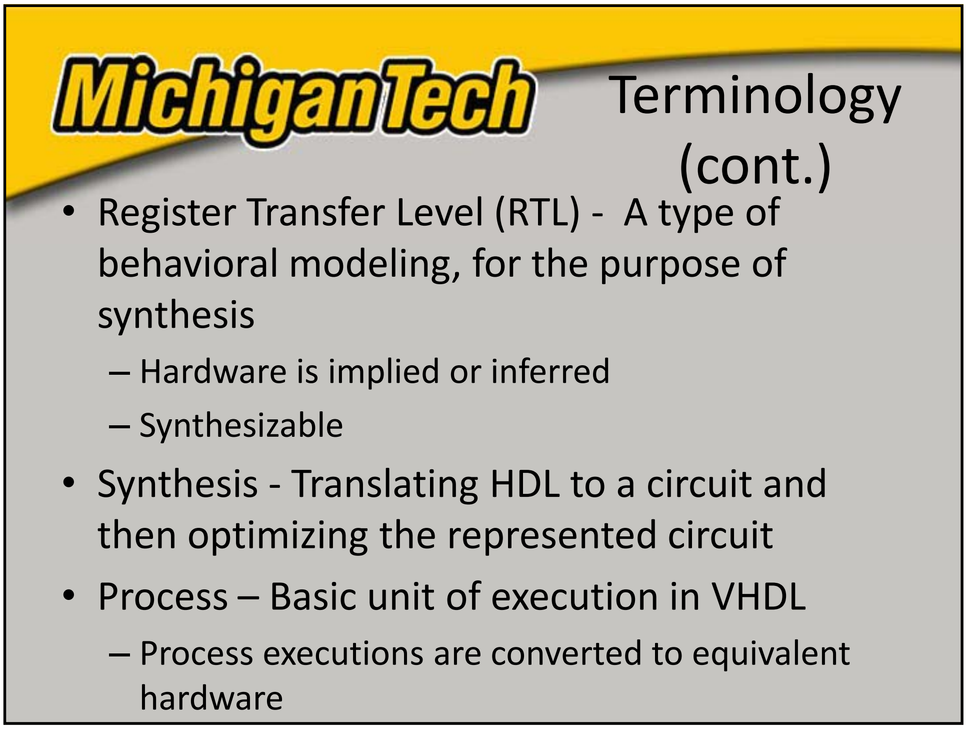
- IEEE industry standard hardware description language
- High-level description language for both simulation & synthesis
  - Facilitates portability and productivity as design logic and verification stimulus are both vendor and tool independent



Terminology

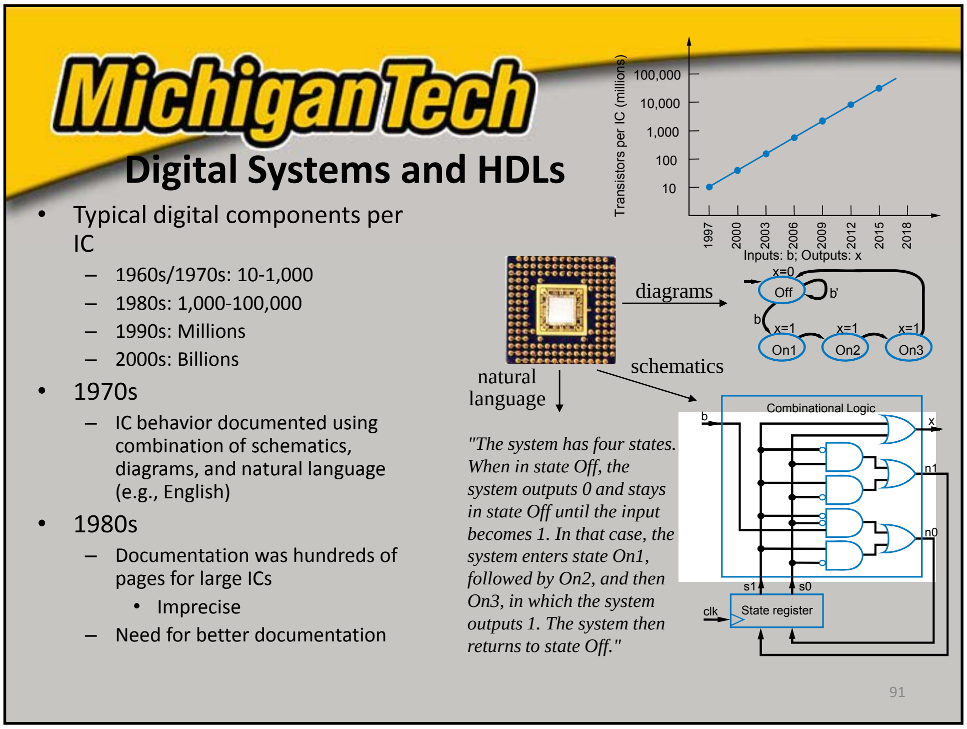
- HDL – A hardware description language is a software programming language that is used to model a piece of hardware
- Behavioral modeling - A component is described by its input/output response
- Structural modeling - A component is described by interconnecting lower-level components/ primitives

89



Terminology (cont.)

- Register Transfer Level (RTL) - A type of behavioral modeling, for the purpose of synthesis
  - Hardware is implied or inferred
  - Synthesizable
- Synthesis - Translating HDL to a circuit and then optimizing the represented circuit
- Process – Basic unit of execution in VHDL
  - Process executions are converted to equivalent hardware



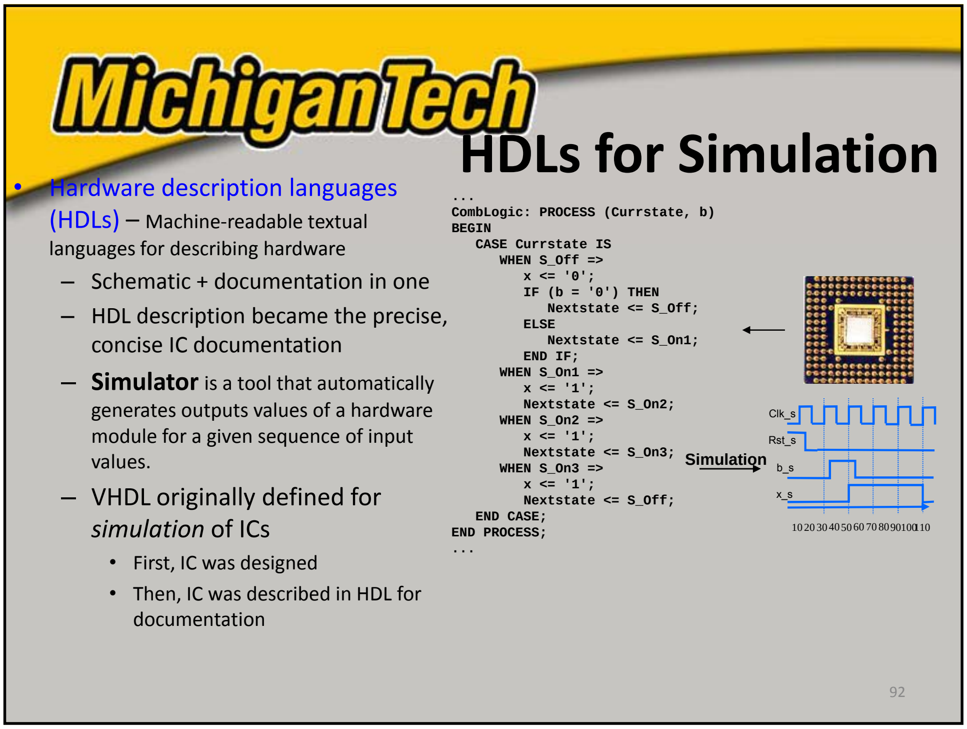
MichiganTech

Digital Systems and HDLs

- Typical digital components per IC
  - 1960s/1970s: 10-1,000
  - 1980s: 1,000-100,000
  - 1990s: Millions
  - 2000s: Billions
- 1970s
  - IC behavior documented using combination of schematics, diagrams, and natural language (e.g., English)
- 1980s
  - Documentation was hundreds of pages for large ICs
    - Imprecise
  - Need for better documentation

"The system has four states. When in state Off, the system outputs 0 and stays in state Off until the input becomes 1. In that case, the system enters state On1, followed by On2, and then On3, in which the system outputs 1. The system then returns to state Off."

91



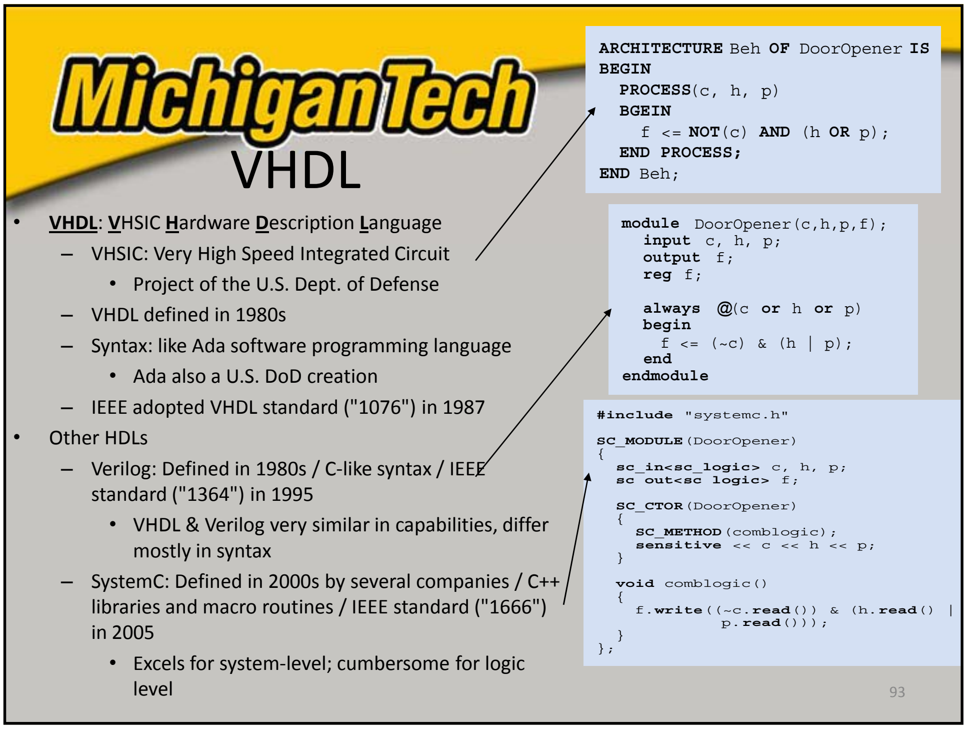
MichiganTech

HDLs for Simulation

- Hardware description languages (HDLs) – Machine-readable textual languages for describing hardware
  - Schematic + documentation in one
  - HDL description became the precise, concise IC documentation
  - **Simulator** is a tool that automatically generates outputs values of a hardware module for a given sequence of input values.
  - VHDL originally defined for *simulation* of ICs
    - First, IC was designed
    - Then, IC was described in HDL for documentation

```
...
CombLogic: PROCESS (Currstate, b)
BEGIN
   CASE Currstate IS
      WHEN S_Off =>
         x <= '0';
         IF (b = '0') THEN
            Nextstate <= S_Off;
         ELSE
            Nextstate <= S_On1;
         END IF;
      WHEN S_On1 =>
         x <= '1';
         Nextstate <= S_On2;
      WHEN S_On2 =>
         x <= '1';
         Nextstate <= S_On3;
      WHEN S_On3 =>
         x <= '1';
         Nextstate <= S_Off;
   END CASE;
END PROCESS;
...
```

92



MichiganTech

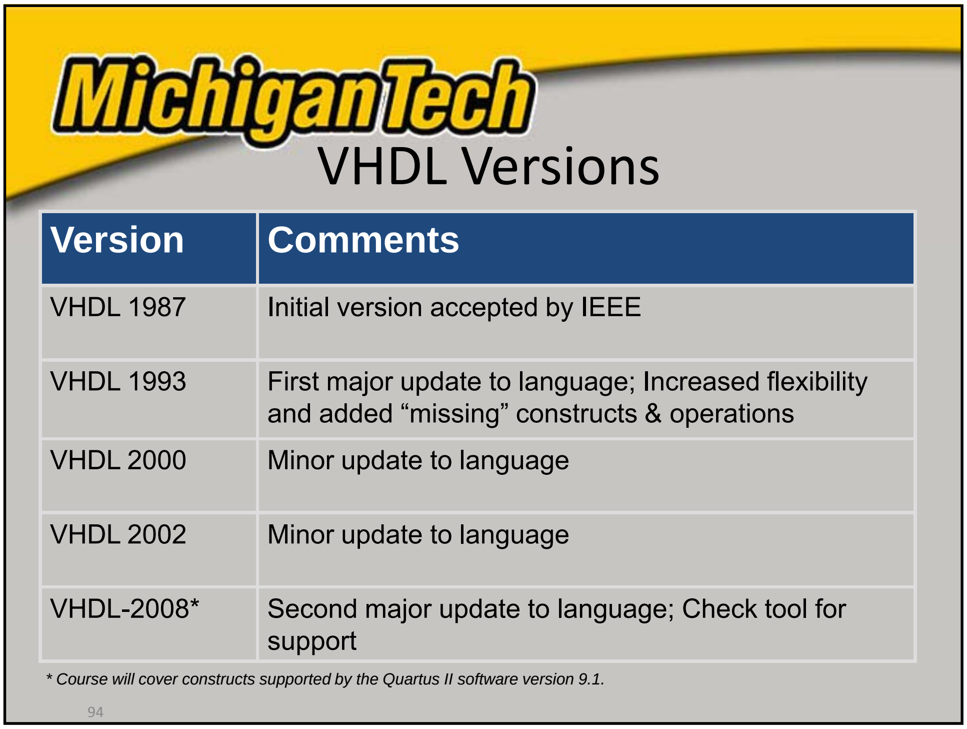
VHDL

- **<u>VHDL</u>**: <u>**V**</u>HSIC <u>**H**</u>ardware <u>**D**</u>escription <u>**L**</u>anguage
  - VHSIC: Very High Speed Integrated Circuit
    - Project of the U.S. Dept. of Defense
  - VHDL defined in 1980s
  - Syntax: like Ada software programming language
    - Ada also a U.S. DoD creation
  - IEEE adopted VHDL standard ("1076") in 1987
- Other HDLs
  - Verilog: Defined in 1980s / C-like syntax / IEEE standard ("1364") in 1995
    - VHDL & Verilog very similar in capabilities, differ mostly in syntax
  - SystemC: Defined in 2000s by several companies / C++ libraries and macro routines / IEEE standard ("1666") in 2005
    - Excels for system-level; cumbersome for logic level

```
ARCHITECTURE Beh OF DoorOpener IS
BEGIN
  PROCESS(c, h, p)
  BGEIN
    f <= NOT(c) AND (h OR p);
  END PROCESS;
END Beh;
```

```
module  DoorOpener(c,h,p,f);
  input  c, h, p;
  output  f;
  reg f;

  always  @(c  or  h  or  p)
  begin
    f  <=  (~c)  &  (h | p);
  end
endmodule
```

```
#include "systemc.h"

SC_MODULE(DoorOpener)
{
  sc_in<sc_logic> c, h, p;
  sc out<sc logic> f;

  SC_CTOR(DoorOpener)
  {
    SC_METHOD(comblogic);
    sensitive << c << h << p;
  }

  void comblogic()
  {
    f.write((~c.read()) & (h.read() |
                p.read()));
  }
};
```

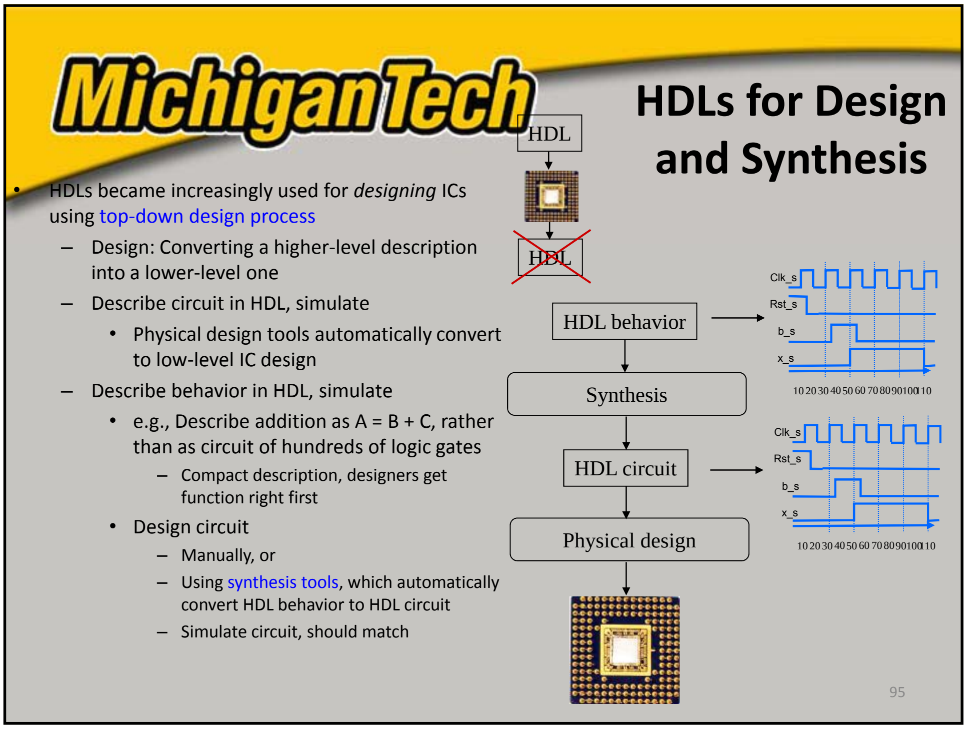


MichiganTech

VHDL Versions

Version	Comments
VHDL 1987	Initial version accepted by IEEE
VHDL 1993	First major update to language; Increased flexibility and added “missing” constructs & operations
VHDL 2000	Minor update to language
VHDL 2002	Minor update to language
VHDL-2008*	Second major update to language; Check tool for support

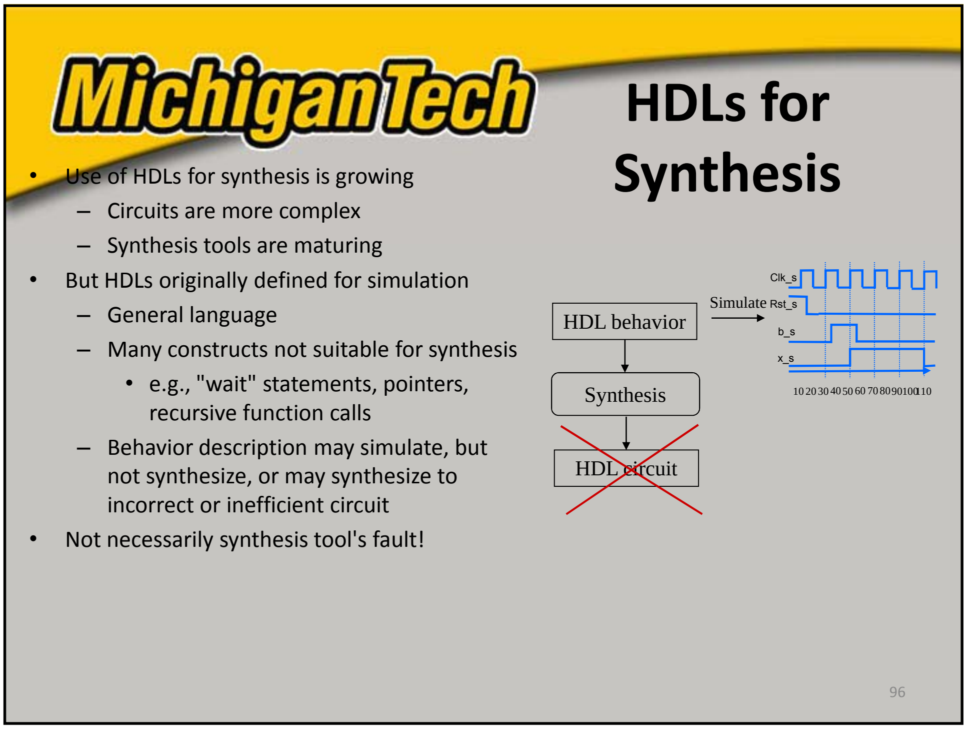
** Course will cover constructs supported by the Quartus II software version 9.1.*



HDLs for Design and Synthesis

- HDLs became increasingly used for *designing* ICs using top-down design process
  - Design: Converting a higher-level description into a lower-level one
  - Describe circuit in HDL, simulate
    - Physical design tools automatically convert to low-level IC design
  - Describe behavior in HDL, simulate
    - e.g., Describe addition as A = B + C, rather than as circuit of hundreds of logic gates
      - Compact description, designers get function right first
    - Design circuit
      - Manually, or
      - Using synthesis tools, which automatically convert HDL behavior to HDL circuit
      - Simulate circuit, should match

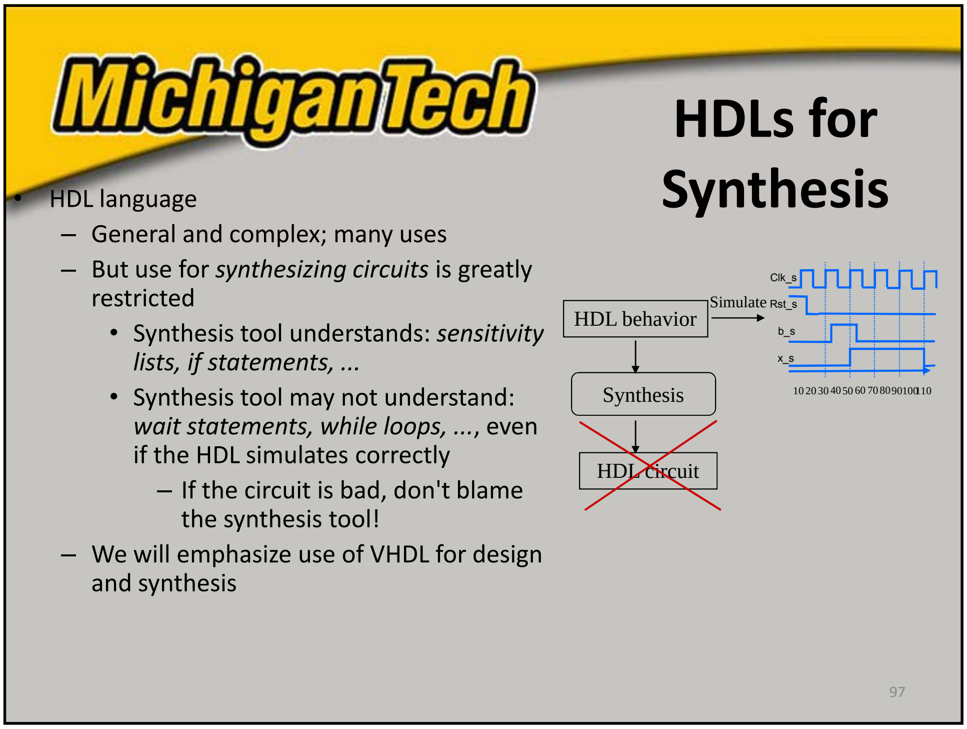
95



HDLs for Synthesis

- Use of HDLs for synthesis is growing
  - Circuits are more complex
  - Synthesis tools are maturing
- But HDLs originally defined for simulation
  - General language
  - Many constructs not suitable for synthesis
    - e.g., "wait" statements, pointers, recursive function calls
  - Behavior description may simulate, but not synthesize, or may synthesize to incorrect or inefficient circuit
- Not necessarily synthesis tool's fault!

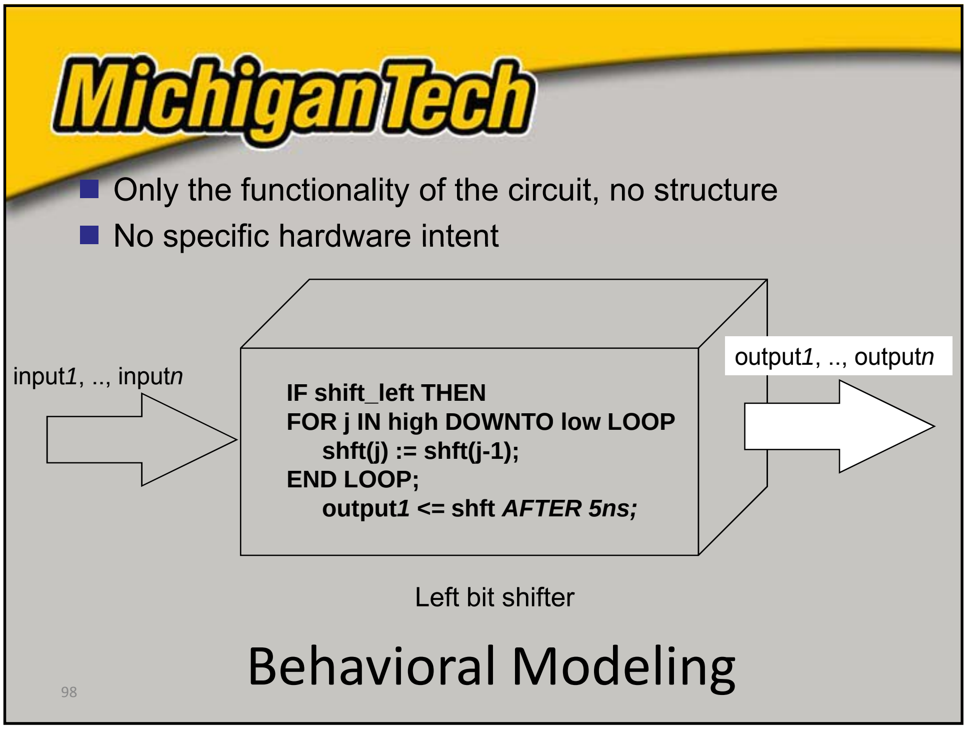
96



HDLs for Synthesis

- HDL language
  - General and complex; many uses
  - But use for *synthesizing circuits* is greatly restricted
    - Synthesis tool understands: *sensitivity lists, if statements, ...*
    - Synthesis tool may not understand: *wait statements, while loops, ...*, even if the HDL simulates correctly
      - If the circuit is bad, don't blame the synthesis tool!
  - We will emphasize use of VHDL for design and synthesis

97



- Only the functionality of the circuit, no structure
- No specific hardware intent

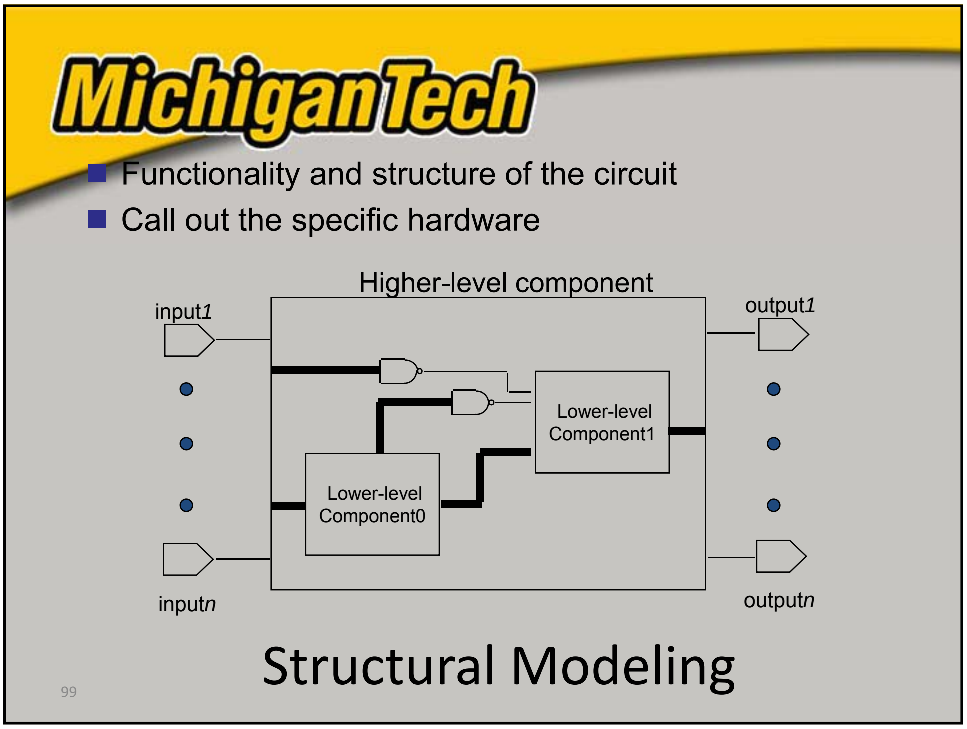
```
IF shift_left THEN
FOR j IN high DOWNTO low LOOP
    shft(j) := shft(j-1);
END LOOP;
    output1 <= shft AFTER 5ns;
```

input*1*, .., input*n*
output*1*, .., output*n*

Left bit shifter

Behavioral Modeling

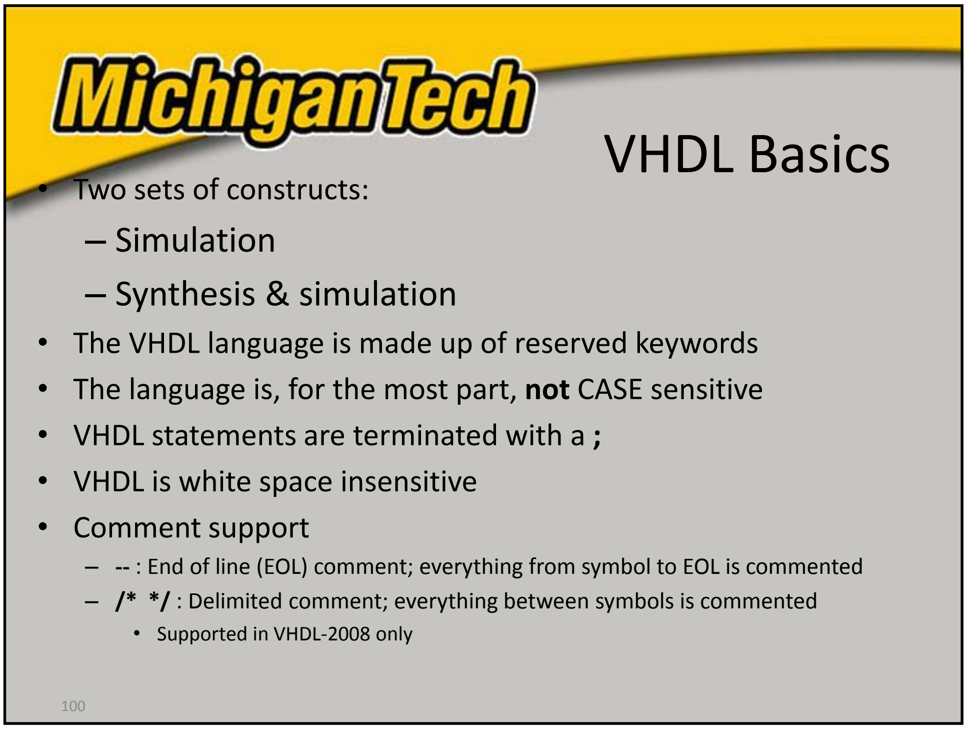
98



- Functionality and structure of the circuit
- Call out the specific hardware

Structural Modeling

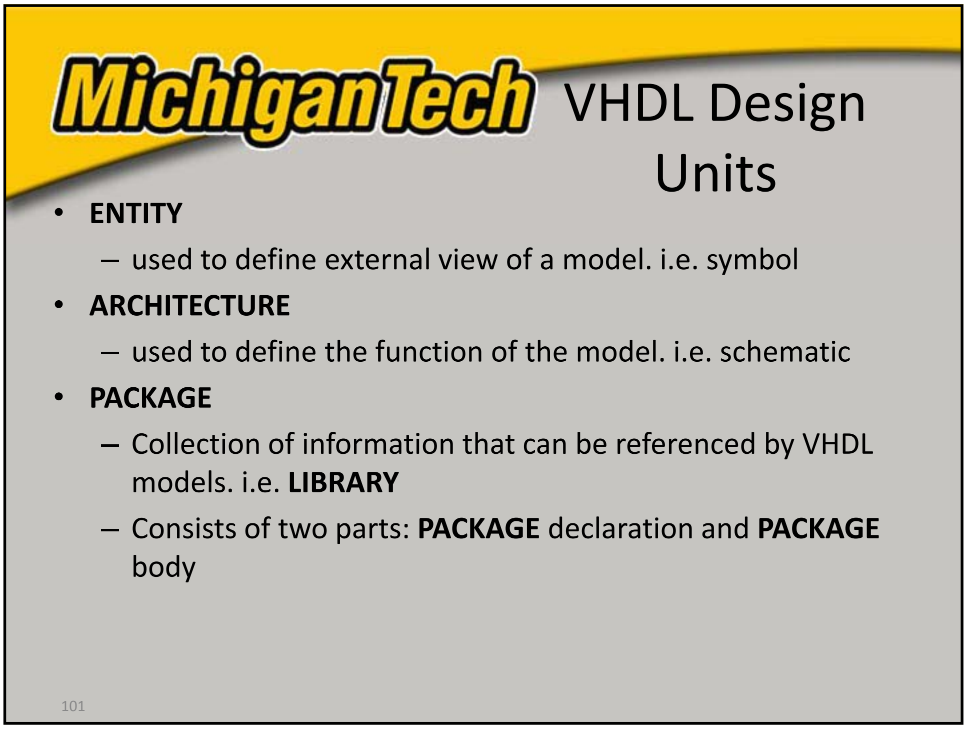
99



VHDL Basics

- Two sets of constructs:
  - Simulation
  - Synthesis & simulation
- The VHDL language is made up of reserved keywords
- The language is, for the most part, **not** CASE sensitive
- VHDL statements are terminated with a **;**
- VHDL is white space insensitive
- Comment support
  - -- : End of line (EOL) comment; everything from symbol to EOL is commented
  - **/* */** : Delimited comment; everything between symbols is commented
    - Supported in VHDL-2008 only

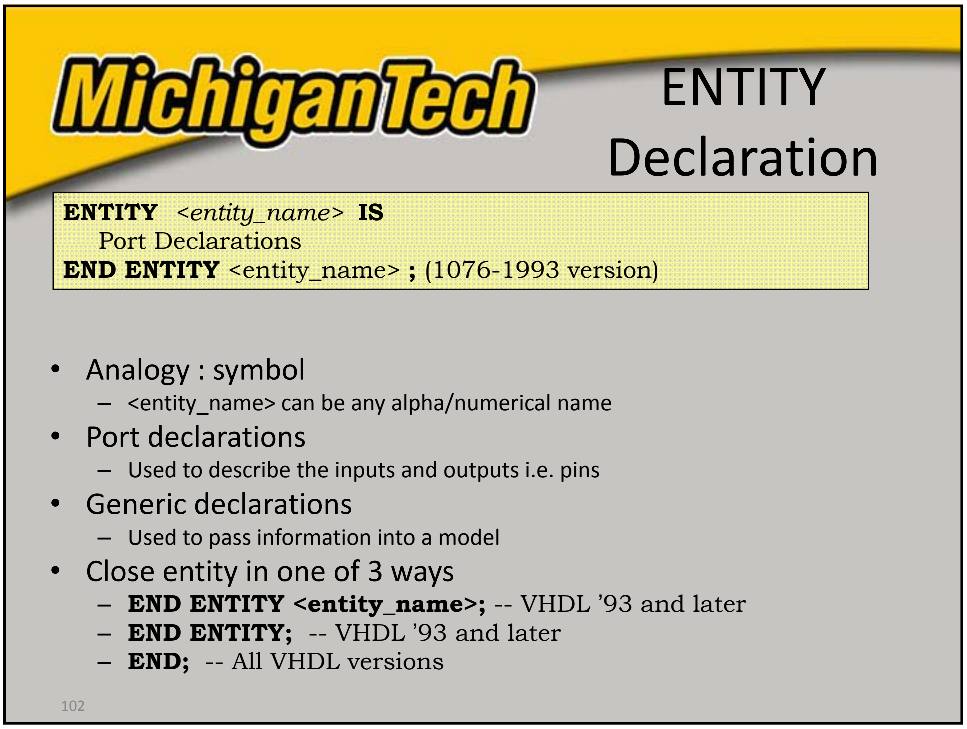
100



VHDL Design Units

- **ENTITY**
  - used to define external view of a model. i.e. symbol
- **ARCHITECTURE**
  - used to define the function of the model. i.e. schematic
- **PACKAGE**
  - Collection of information that can be referenced by VHDL models. i.e. **LIBRARY**
  - Consists of two parts: **PACKAGE** declaration and **PACKAGE** body

101

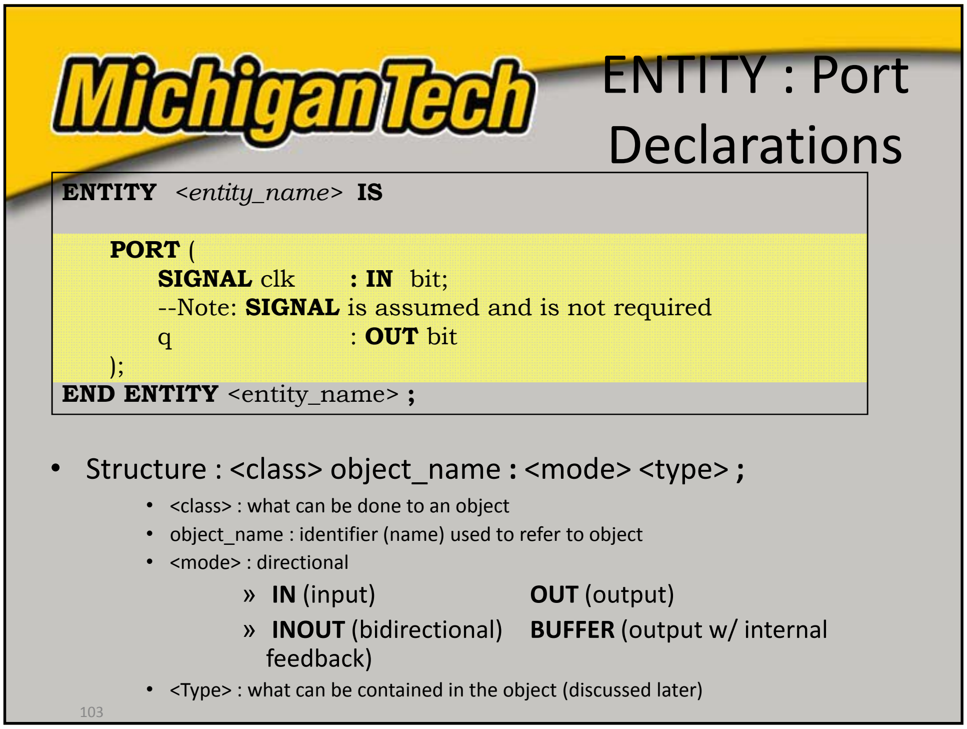


ENTITY Declaration

```
ENTITY  <entity_name>  IS
   Port Declarations
END ENTITY <entity_name> ; (1076-1993 version)
```

- Analogy : symbol
  - <entity_name> can be any alpha/numerical name
- Port declarations
  - Used to describe the inputs and outputs i.e. pins
- Generic declarations
  - Used to pass information into a model
- Close entity in one of 3 ways
  - **END ENTITY <entity_name>;** -- VHDL ’93 and later
  - **END ENTITY;** -- VHDL ’93 and later
  - **END;** -- All VHDL versions

102

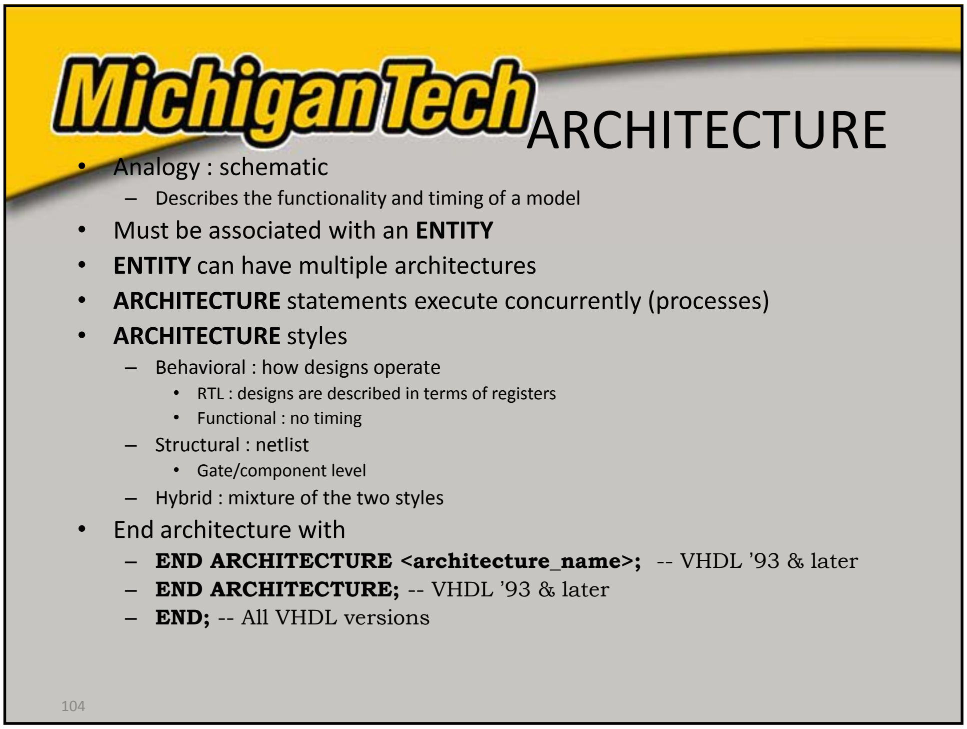


ENTITY : Port Declarations

```
ENTITY  <entity_name>  IS

     PORT (
          SIGNAL clk        : IN  bit;
          --Note: SIGNAL is assumed and is not required
          q                 : OUT bit
     );
END ENTITY <entity_name> ;
```

- Structure : <class> object_name **:** <mode> <type> **;**
  - <class> : what can be done to an object
  - object_name : identifier (name) used to refer to object
  - <mode> : directional
    - » **IN** (input) **OUT** (output)
    - » **INOUT** (bidirectional) **BUFFER** (output w/ internal feedback)
  - <Type> : what can be contained in the object (discussed later)

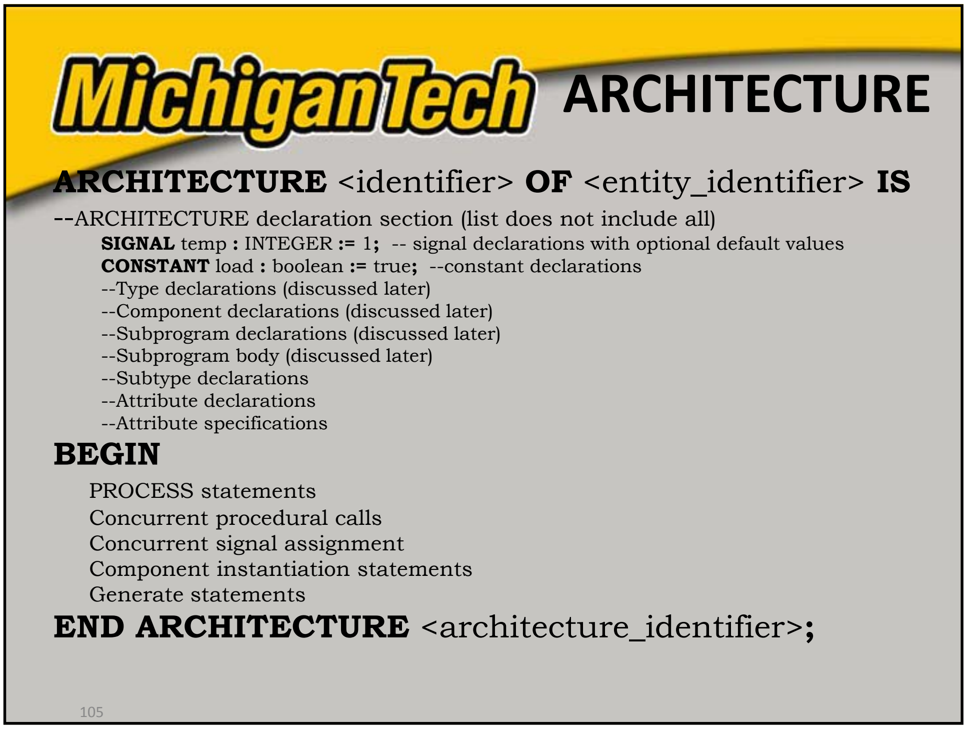
103



ARCHITECTURE

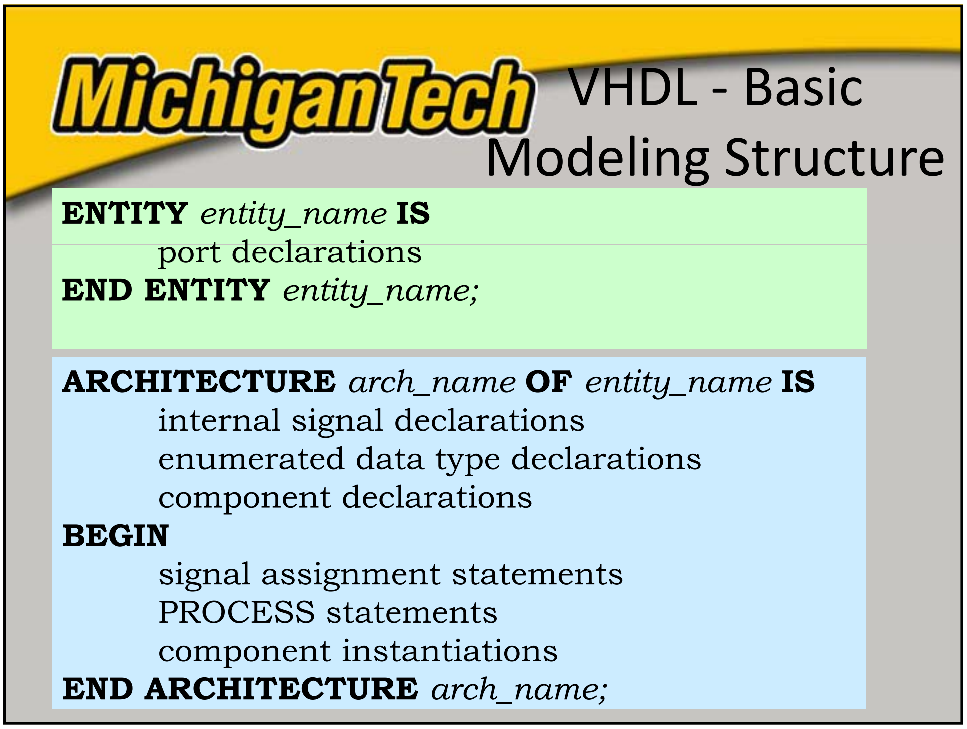
- Analogy : schematic
  - Describes the functionality and timing of a model
- Must be associated with an **ENTITY**
- **ENTITY** can have multiple architectures
- **ARCHITECTURE** statements execute concurrently (processes)
- **ARCHITECTURE** styles
  - Behavioral : how designs operate
    - RTL : designs are described in terms of registers
    - Functional : no timing
  - Structural : netlist
    - Gate/component level
  - Hybrid : mixture of the two styles
- End architecture with
  - **END ARCHITECTURE <architecture_name>;** -- VHDL '93 & later
  - **END ARCHITECTURE;** -- VHDL '93 & later
  - **END;** -- All VHDL versions

104



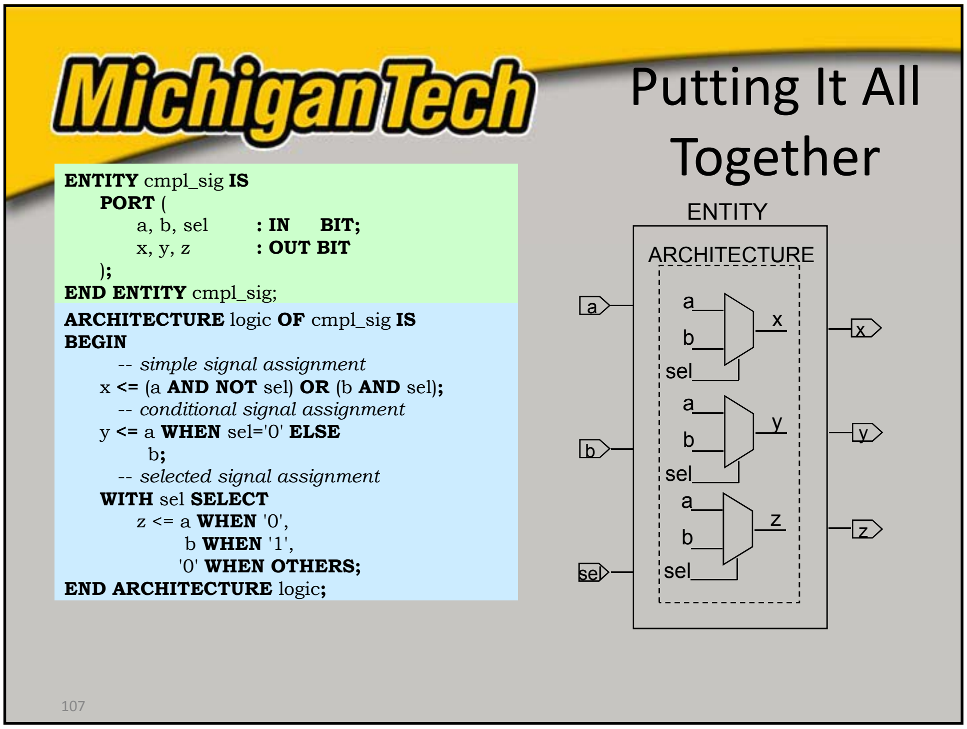
ARCHITECTURE

```
ARCHITECTURE <identifier> OF <entity_identifier> IS
--ARCHITECTURE declaration section (list does not include all)
    SIGNAL temp : INTEGER := 1;  -- signal declarations with optional default values
    CONSTANT load : boolean := true;  --constant declarations
    --Type declarations (discussed later)
    --Component declarations (discussed later)
    --Subprogram declarations (discussed later)
    --Subprogram body (discussed later)
    --Subtype declarations
    --Attribute declarations
    --Attribute specifications
BEGIN
   PROCESS statements
   Concurrent procedural calls
   Concurrent signal assignment
   Component instantiation statements
   Generate statements
END ARCHITECTURE <architecture_identifier>;
```



VHDL - Basic Modeling Structure

```
ENTITY entity_name IS
        port declarations
END ENTITY entity_name;

ARCHITECTURE arch_name OF entity_name IS
        internal signal declarations
        enumerated data type declarations
        component declarations
BEGIN
        signal assignment statements
        PROCESS statements
        component instantiations
END ARCHITECTURE arch_name;
```

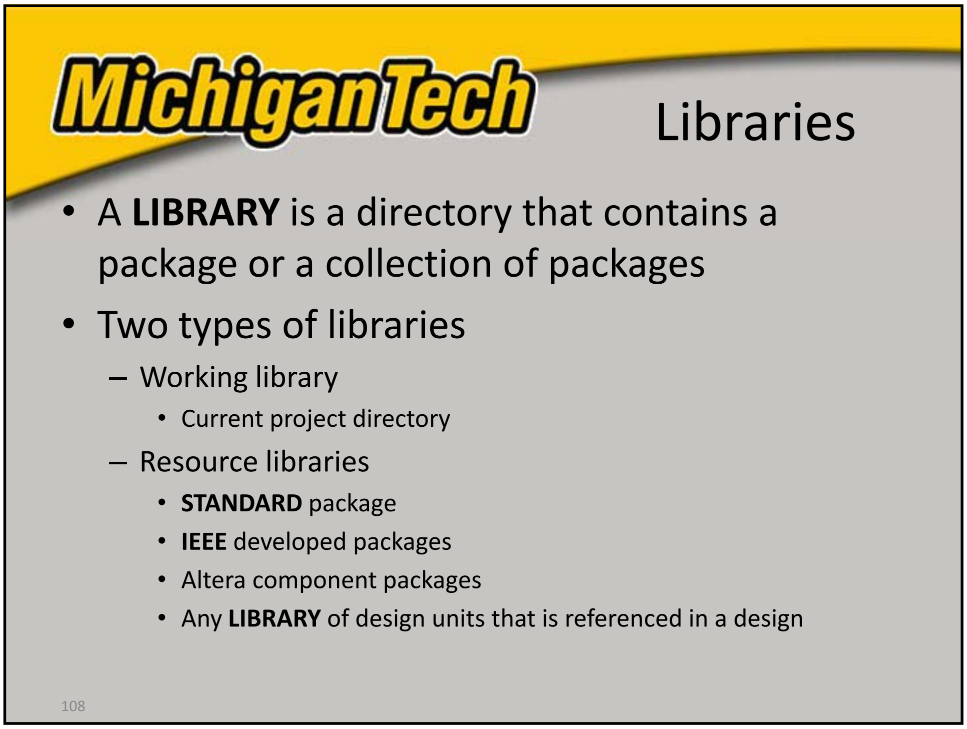


Putting It All Together

```
ENTITY cmpl_sig IS
     PORT (
          a, b, sel        : IN     BIT;
          x, y, z           : OUT BIT
     );
END ENTITY cmpl_sig;
ARCHITECTURE logic OF cmpl_sig IS
BEGIN
        -- simple signal assignment
     x <= (a AND NOT sel) OR (b AND sel);
        -- conditional signal assignment
     y <= a WHEN sel='0' ELSE
          b;
        -- selected signal assignment
     WITH sel SELECT
          z <= a WHEN '0',
               b WHEN '1',
              '0' WHEN OTHERS;
END ARCHITECTURE logic;
```

ENTITY
ARCHITECTURE
a
b
sel
a
b
sel
x
a
b
sel
y
a
b
sel
z
x
y
z

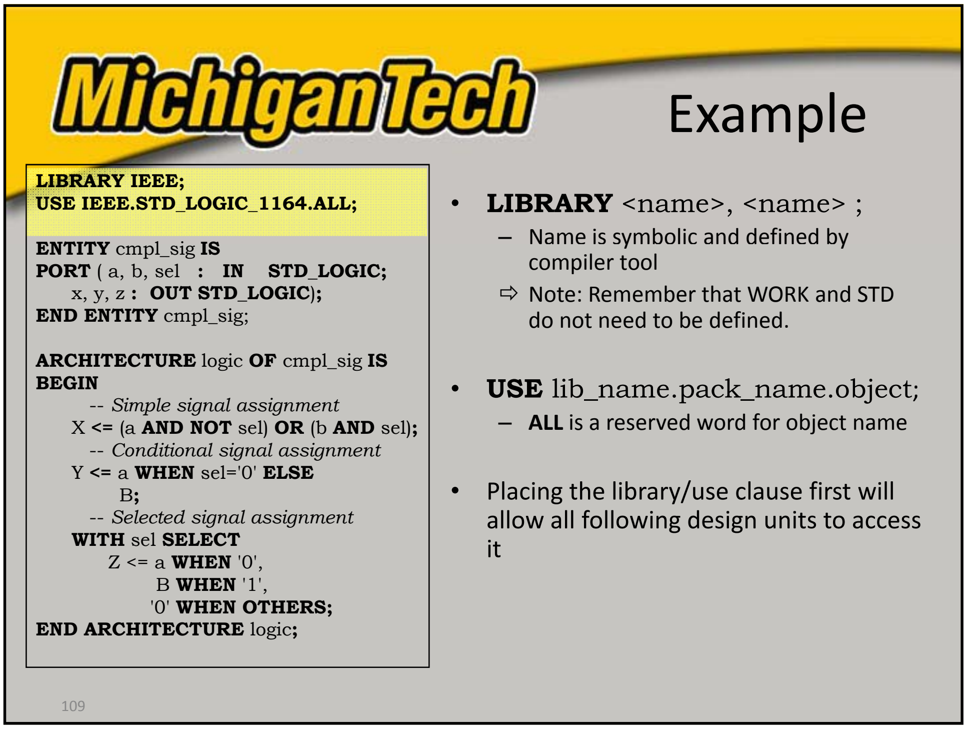
107



Libraries

- A **LIBRARY** is a directory that contains a package or a collection of packages
- Two types of libraries
  - Working library
    - Current project directory
  - Resource libraries
    - **STANDARD** package
    - **IEEE** developed packages
    - Altera component packages
    - Any **LIBRARY** of design units that is referenced in a design

108

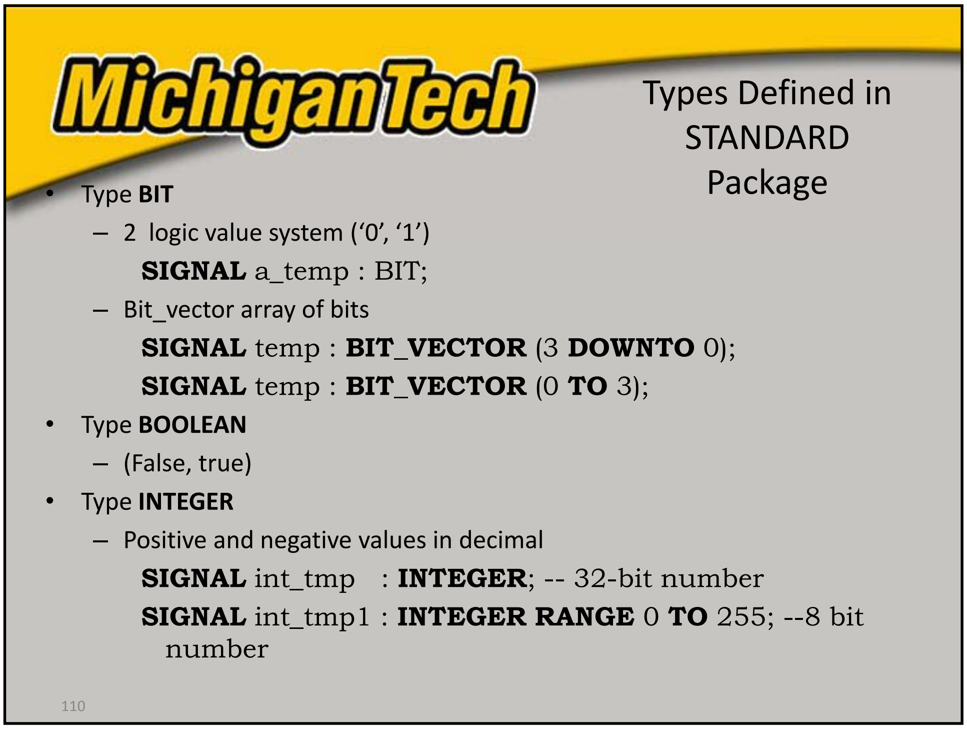


Example

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cmpl_sig IS
PORT ( a, b, sel   :   IN     STD_LOGIC;
     x, y, z :  OUT STD_LOGIC);
END ENTITY cmpl_sig;

ARCHITECTURE logic OF cmpl_sig IS
BEGIN
        -- Simple signal assignment
     X <= (a AND NOT sel) OR (b AND sel);
        -- Conditional signal assignment
     Y <= a WHEN sel='0' ELSE
            B;
        -- Selected signal assignment
     WITH sel SELECT
          Z <= a WHEN '0',
                 B WHEN '1',
                '0' WHEN OTHERS;
END ARCHITECTURE logic;
```

- **LIBRARY** <name>, <name> ;
  - Name is symbolic and defined by compiler tool
  - ⇨ Note: Remember that WORK and STD do not need to be defined.
- **USE** lib_name.pack_name.object;
  - **ALL** is a reserved word for object name
- Placing the library/use clause first will allow all following design units to access it

109



Types Defined in STANDARD Package

- Type **BIT**
  - 2 logic value system ('0', '1')

    SIGNAL a_temp : BIT;
  - Bit_vector array of bits

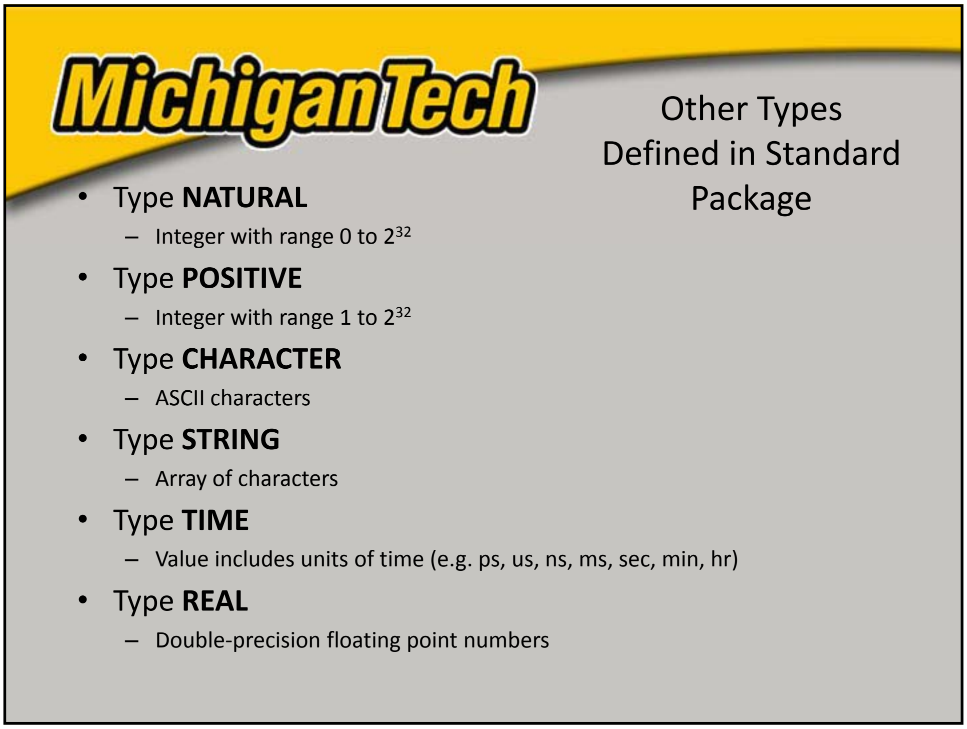
    SIGNAL temp : **BIT_VECTOR** (3 **DOWNTO** 0);

    SIGNAL temp : **BIT_VECTOR** (0 **TO** 3);
- Type **BOOLEAN**
  - (False, true)
- Type **INTEGER**
  - Positive and negative values in decimal

    SIGNAL int_tmp : **INTEGER**; -- 32-bit number

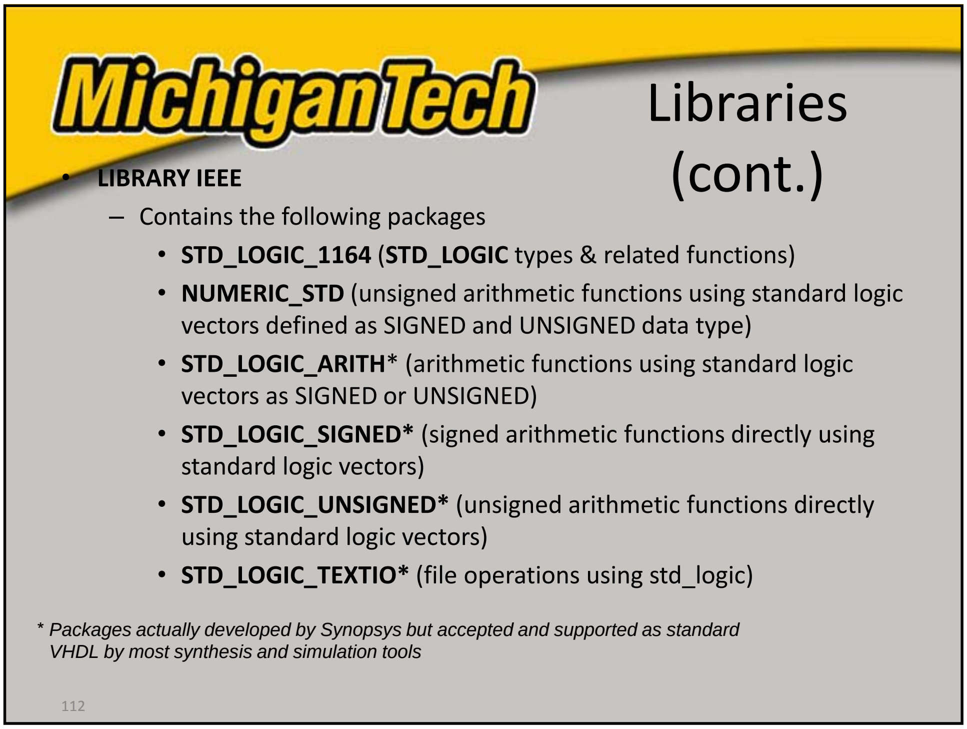
    SIGNAL int_tmp1 : **INTEGER RANGE** 0 **TO** 255; --8 bit number

110



Other Types Defined in Standard Package

- Type **NATURAL**
  - Integer with range 0 to 2^{32}
- Type **POSITIVE**
  - Integer with range 1 to 2^{32}
- Type **CHARACTER**
  - ASCII characters
- Type **STRING**
  - Array of characters
- Type **TIME**
  - Value includes units of time (e.g. ps, us, ns, ms, sec, min, hr)
- Type **REAL**
  - Double-precision floating point numbers

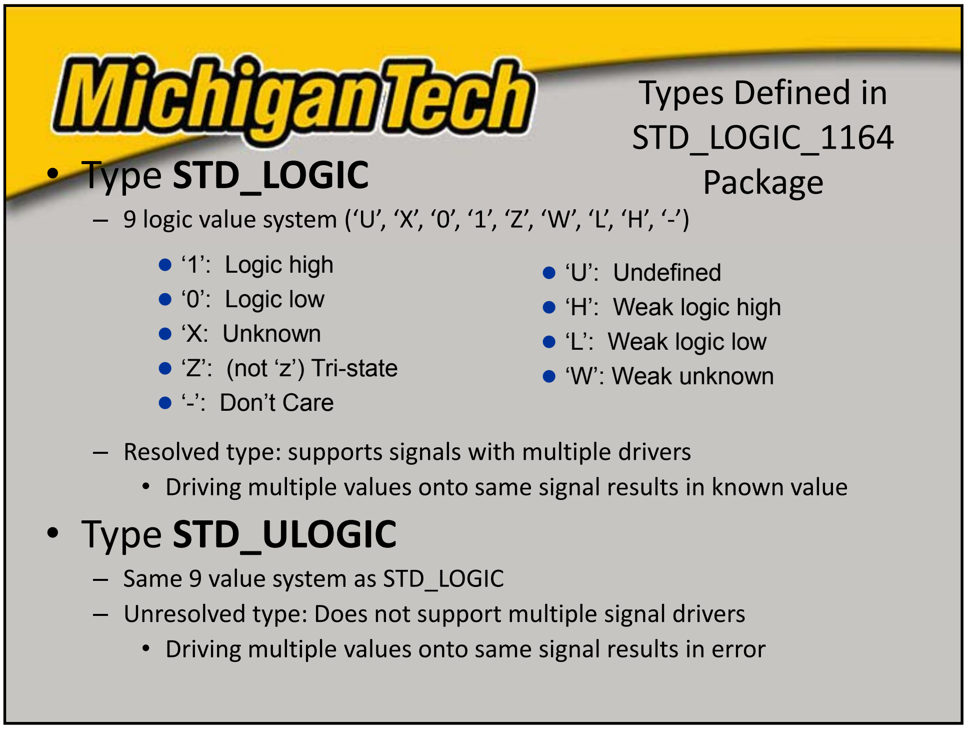


Libraries (cont.)

- **LIBRARY IEEE**
  - Contains the following packages
    - **STD_LOGIC_1164** (**STD_LOGIC** types & related functions)
    - **NUMERIC_STD** (unsigned arithmetic functions using standard logic vectors defined as SIGNED and UNSIGNED data type)
    - **STD_LOGIC_ARITH*** (arithmetic functions using standard logic vectors as SIGNED or UNSIGNED)
    - **STD_LOGIC_SIGNED*** (signed arithmetic functions directly using standard logic vectors)
    - **STD_LOGIC_UNSIGNED*** (unsigned arithmetic functions directly using standard logic vectors)
    - **STD_LOGIC_TEXTIO*** (file operations using std_logic)

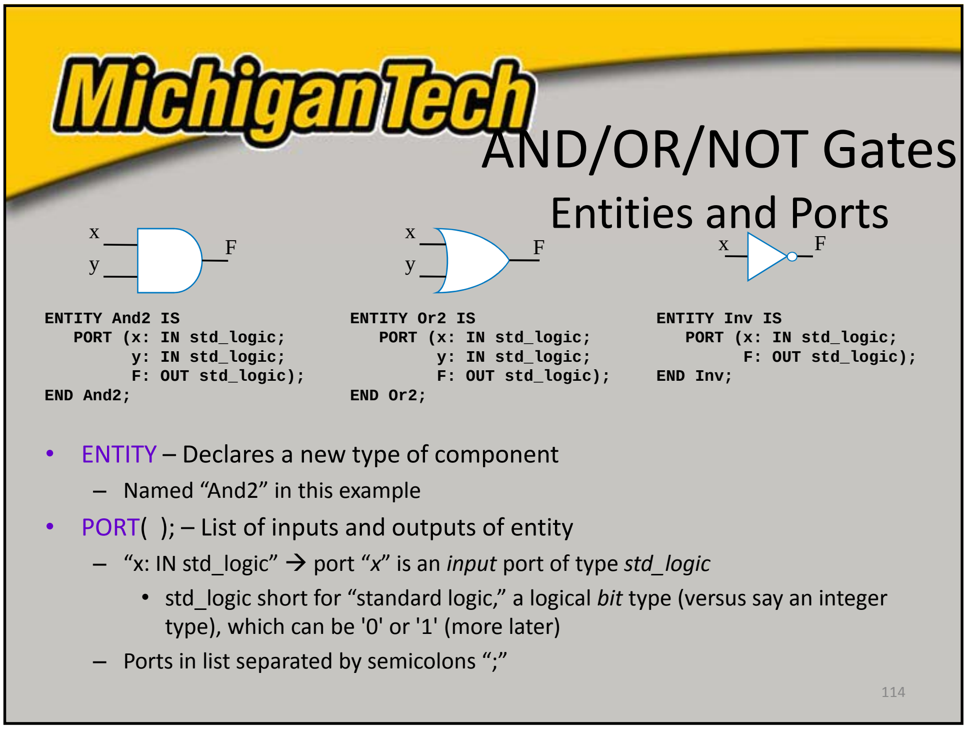
** Packages actually developed by Synopsys but accepted and supported as standard VHDL by most synthesis and simulation tools*

112



Types Defined in STD_LOGIC_1164 Package

- Type **STD_LOGIC**
  - 9 logic value system (‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’)
    - ‘1’: Logic high
    - ‘0’: Logic low
    - ‘X: Unknown
    - ‘Z’: (not ‘z’) Tri-state
    - ‘-’: Don’t Care
    - ‘U’: Undefined
    - ‘H’: Weak logic high
    - ‘L’: Weak logic low
    - ‘W’: Weak unknown
  - Resolved type: supports signals with multiple drivers
    - Driving multiple values onto same signal results in known value
- Type **STD_ULOGIC**
  - Same 9 value system as STD_LOGIC
  - Unresolved type: Does not support multiple signal drivers
    - Driving multiple values onto same signal results in error



AND/OR/NOT Gates

Entities and Ports

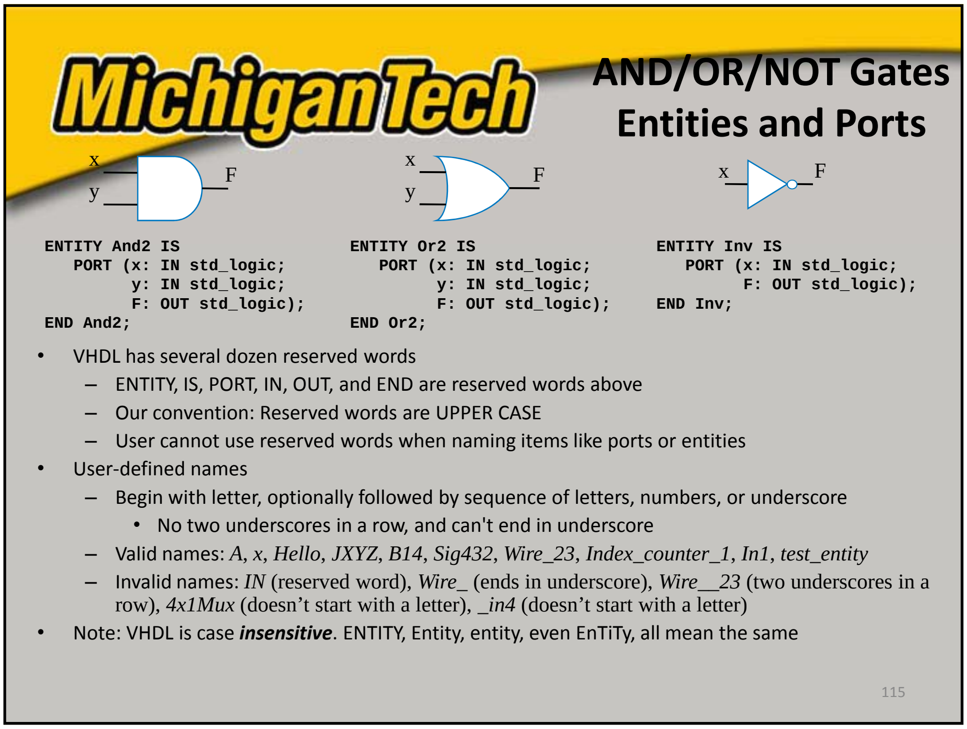
```
ENTITY And2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END And2;
```

```
ENTITY Or2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END Or2;
```

```
ENTITY Inv IS
   PORT (x: IN std_logic;
         F: OUT std_logic);
END Inv;
```

- ENTITY – Declares a new type of component
  - Named “And2” in this example
- PORT( ); – List of inputs and outputs of entity
  - “x: IN std_logic” → port “x” is an *input* port of type *std_logic*
    - std_logic short for “standard logic,” a logical *bit* type (versus say an integer type), which can be '0' or '1' (more later)
  - Ports in list separated by semicolons “;”

114



MichiganTech
x
y
F
x
y
F
x
F

AND/OR/NOT Gates Entities and Ports

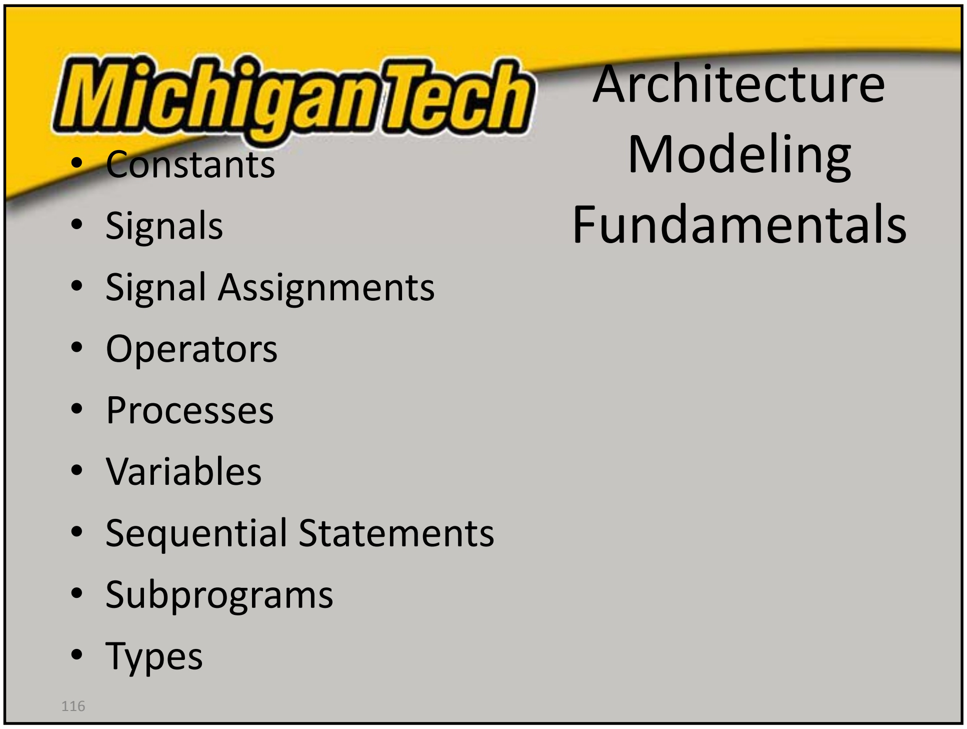
```
ENTITY And2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END And2;
```

```
ENTITY Or2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END Or2;
```

```
ENTITY Inv IS
   PORT (x: IN std_logic;
         F: OUT std_logic);
END Inv;
```

- VHDL has several dozen reserved words
  - ENTITY, IS, PORT, IN, OUT, and END are reserved words above
  - Our convention: Reserved words are UPPER CASE
  - User cannot use reserved words when naming items like ports or entities
- User-defined names
  - Begin with letter, optionally followed by sequence of letters, numbers, or underscore
    - No two underscores in a row, and can't end in underscore
  - Valid names: *A*, *x*, *Hello*, *JXYZ*, *B14*, *Sig432*, *Wire_23*, *Index_counter_1*, *In1*, *test_entity*
  - Invalid names: *IN* (reserved word), *Wire_* (ends in underscore), *Wire__23* (two underscores in a row), *4x1Mux* (doesn’t start with a letter), *_in4* (doesn’t start with a letter)
- Note: VHDL is case ***insensitive***. ENTITY, Entity, entity, even EnTiTy, all mean the same

115

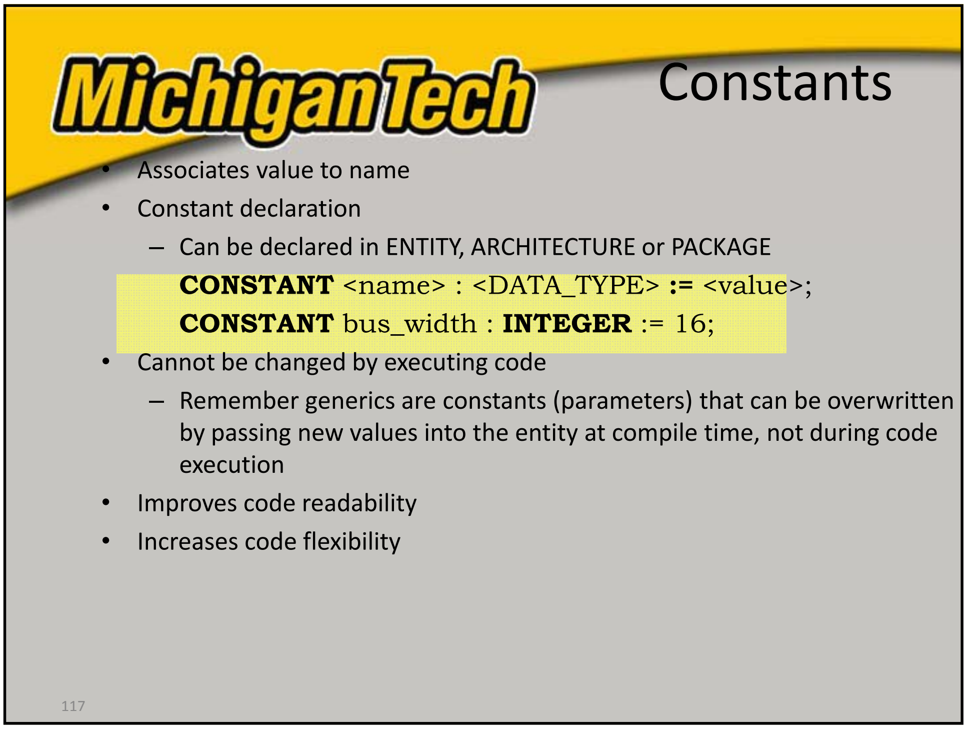


MichiganTech

Architecture Modeling Fundamentals

- Constants
- Signals
- Signal Assignments
- Operators
- Processes
- Variables
- Sequential Statements
- Subprograms
- Types

116



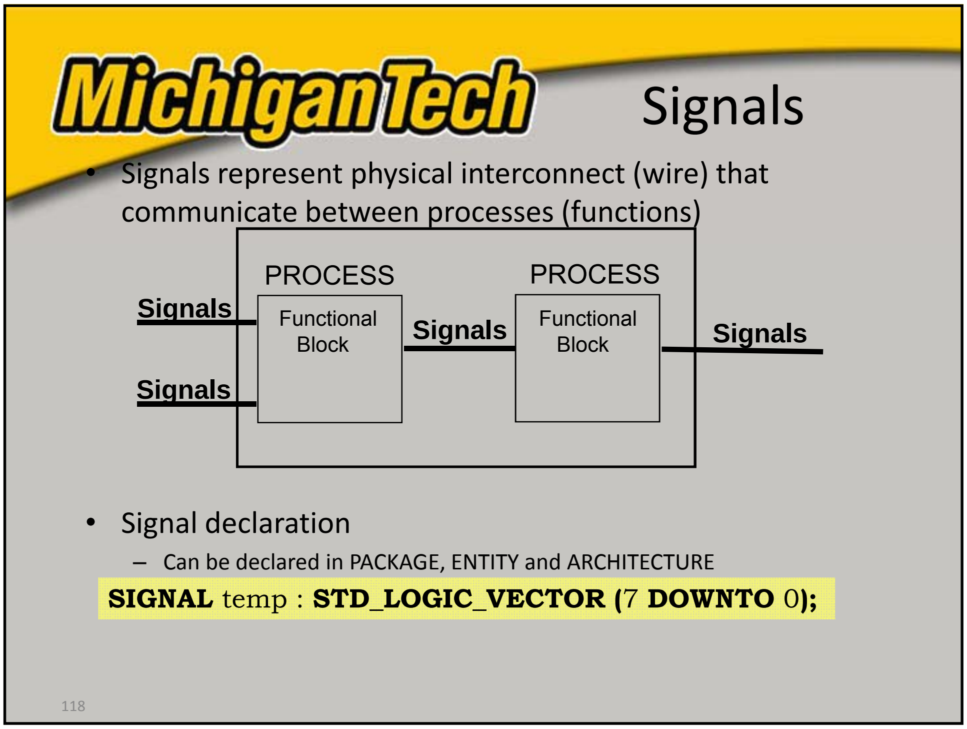
Constants

- Associates value to name
- Constant declaration
  - Can be declared in ENTITY, ARCHITECTURE or PACKAGE

```
CONSTANT <name> : <DATA_TYPE> := <value>;
CONSTANT bus_width : INTEGER := 16;
```

- Cannot be changed by executing code
  - Remember generics are constants (parameters) that can be overwritten by passing new values into the entity at compile time, not during code execution
- Improves code readability
- Increases code flexibility

117



Signals

- Signals represent physical interconnect (wire) that communicate between processes (functions)

- Signal declaration
  - Can be declared in PACKAGE, ENTITY and ARCHITECTURE

```
SIGNAL temp : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

118

Assigning Values to Signals

SIGNAL temp : **STD_LOGIC_VECTOR** (7 **DOWNTO** 0);

- Signal assignments are represented by **<=**
- Examples
  - All bits

    temp <= "10101010";

    temp <= x"aa" ; (1076-1993)
    - VHDL also supports 'o' for octal and 'b' for binary
  - Bit-slicing

    temp (7 **DOWNTO** 4) <= "1010";
  - Single bit

    temp(7) <= '1';

- Use double-quotes (" ") to assign multi-bit values and single-quotes (' ') to assign single-bit values

119

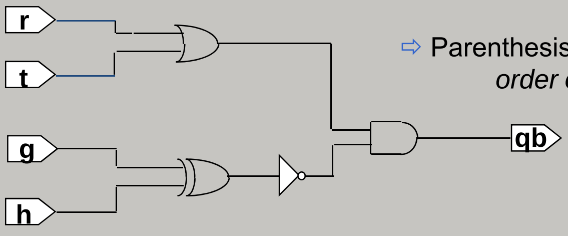
Simple Signal Assignments

- Format: *<signal_name>* **<=** *<expression>***;**

- Example:

```
qa <=   r OR t ;
qb <=   (qa AND NOT (g XOR h));
```

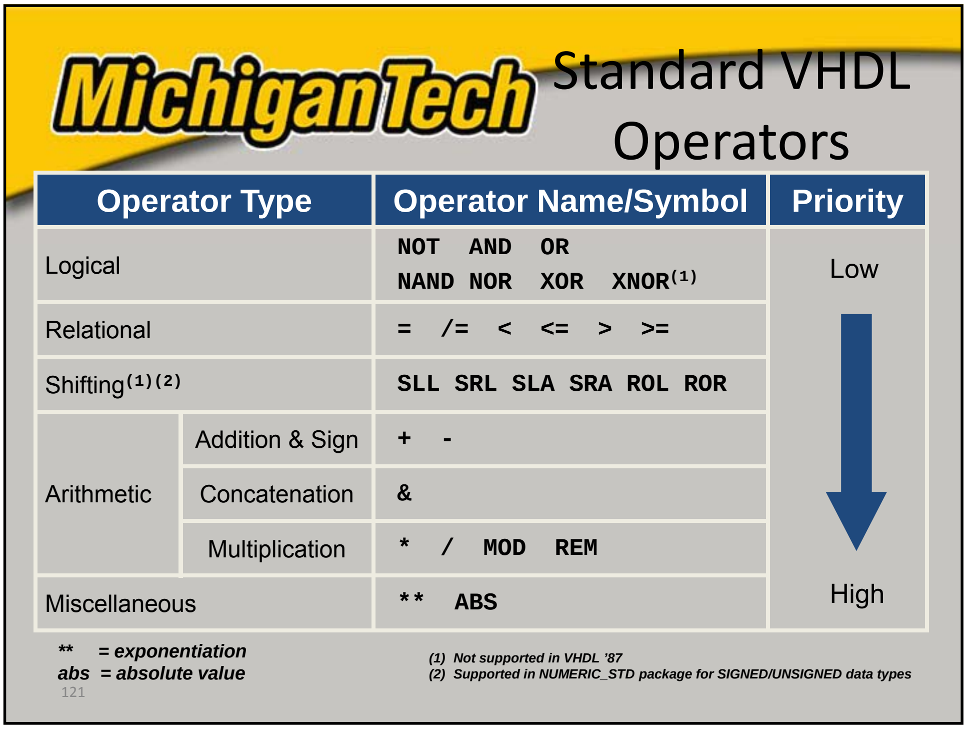
2 implied processes



⇨ Parenthesis **( )** *give the order of operation*

- Expressions use VHDL operators to describe behavior

120



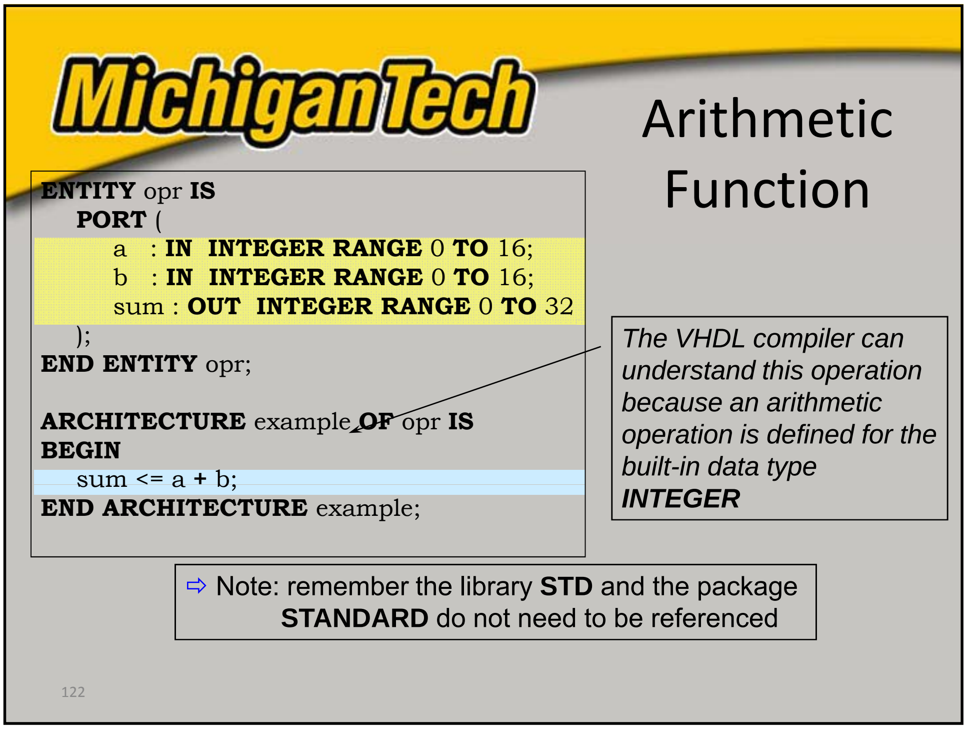
Standard VHDL Operators

Operator Type		Operator Name/Symbol	Priority
Logical		NOT AND OR NAND NOR XOR XNOR(1)	Low
Relational		= /= < <= > >=	
Shifting(1)(2)		SLL SRL SLA SRA ROL ROR	
Arithmetic	Addition & Sign	+ -	
	Concatenation	&	
	Multiplication	* / MOD REM	
Miscellaneous		** ABS	High

*** = exponentiation*
abs = absolute value

(1) Not supported in VHDL '87
(2) Supported in NUMERIC_STD package for SIGNED/UNSIGNED data types

121



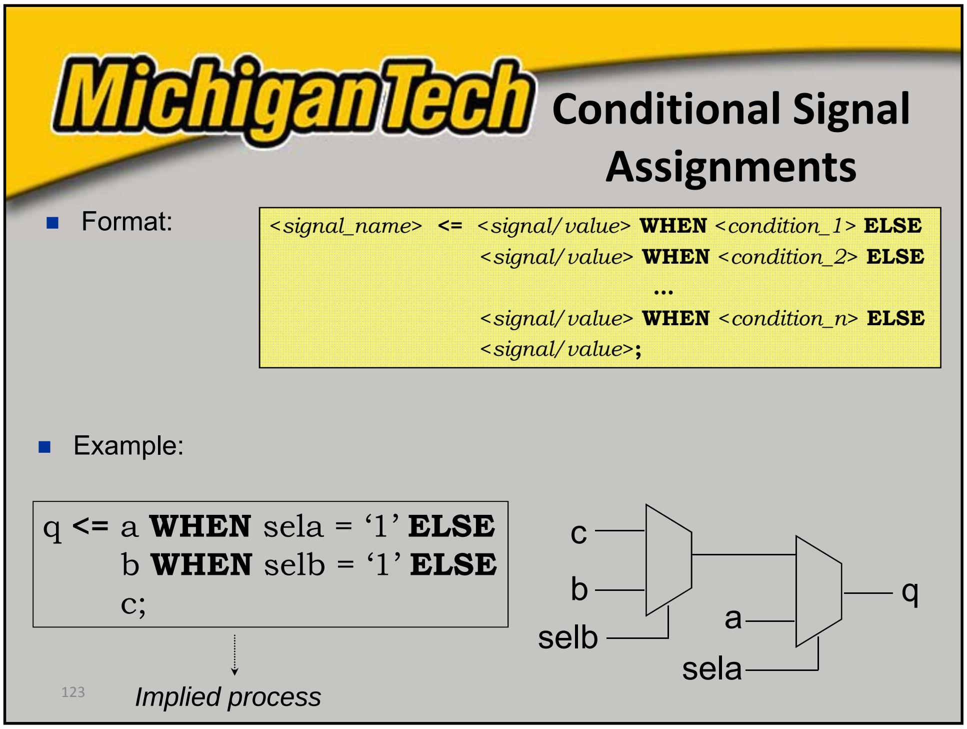
Arithmetic Function

```
ENTITY opr IS
   PORT (
      a   : IN  INTEGER RANGE 0 TO 16;
      b   : IN  INTEGER RANGE 0 TO 16;
      sum : OUT  INTEGER RANGE 0 TO 32
   );
END ENTITY opr;

ARCHITECTURE example OF opr IS
BEGIN
   sum <= a + b;
END ARCHITECTURE example;
```

The VHDL compiler can understand this operation because an arithmetic operation is defined for the built-in data type ***INTEGER***

⇨ Note: remember the library **STD** and the package **STANDARD** do not need to be referenced

122



Conditional Signal Assignments

- Format:

```
<signal_name>  <=  <signal/value> WHEN <condition_1> ELSE
                   <signal/value> WHEN <condition_2> ELSE
                                   …
                   <signal/value> WHEN <condition_n> ELSE
                   <signal/value>;
```

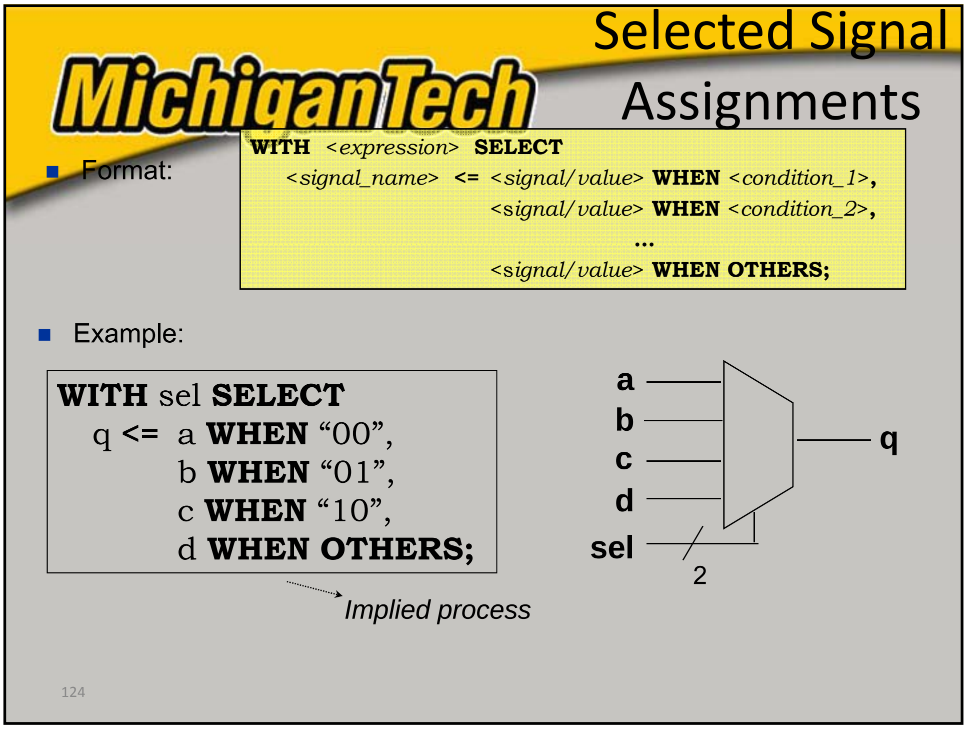
- Example:

```
q <= a WHEN sela = '1' ELSE
     b WHEN selb = '1' ELSE
     c;
```

Implied process

c
b
selb
a
sela
q

123



Selected Signal Assignments

- Format:

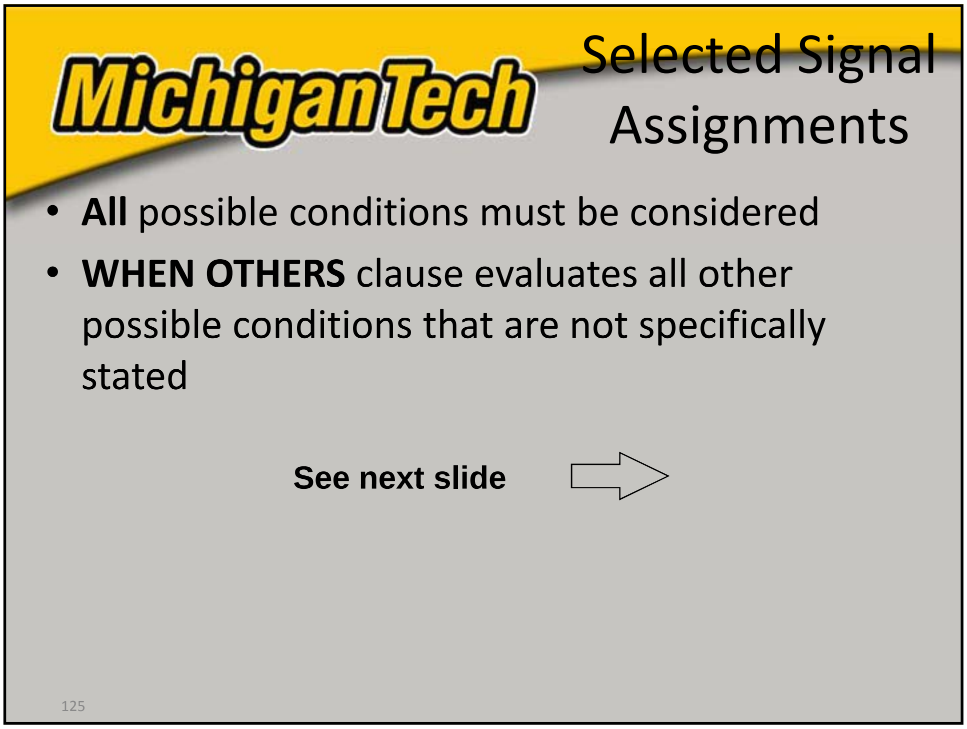
```
WITH  <expression>  SELECT
   <signal_name>  <=  <signal/value> WHEN <condition_1>,
                      <signal/value> WHEN <condition_2>,
                                      …
                      <signal/value> WHEN OTHERS;
```

- Example:

```
WITH sel SELECT
   q <=  a WHEN "00",
         b WHEN "01",
         c WHEN "10",
         d WHEN OTHERS;
```

Implied process

124

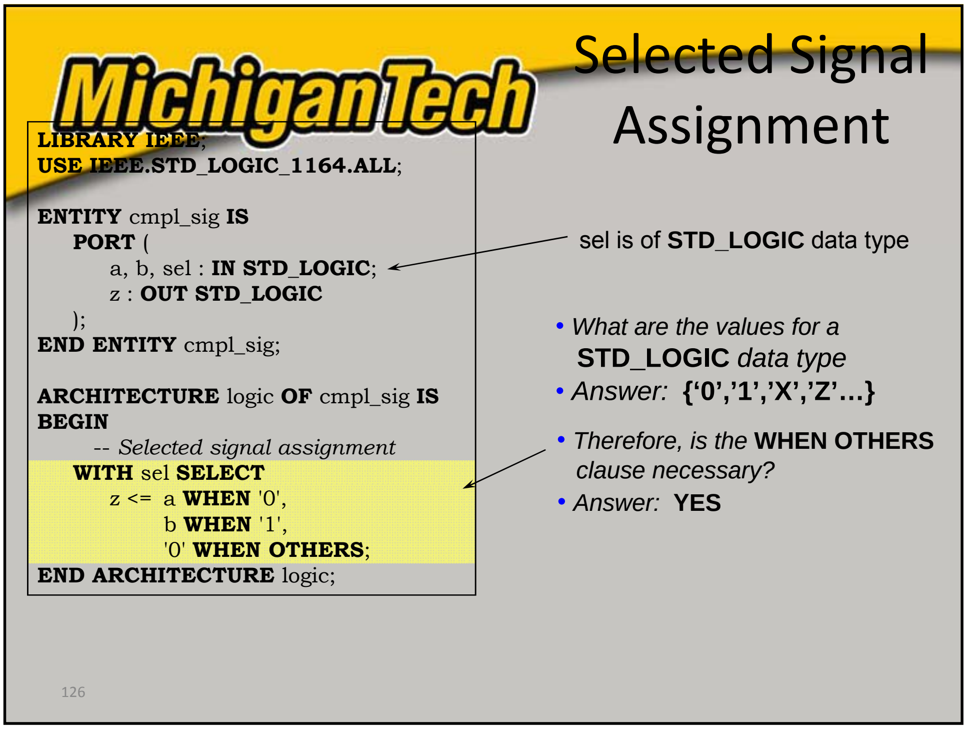


Selected Signal Assignments

- **All** possible conditions must be considered
- **WHEN OTHERS** clause evaluates all other possible conditions that are not specifically stated

See next slide ⇨

125



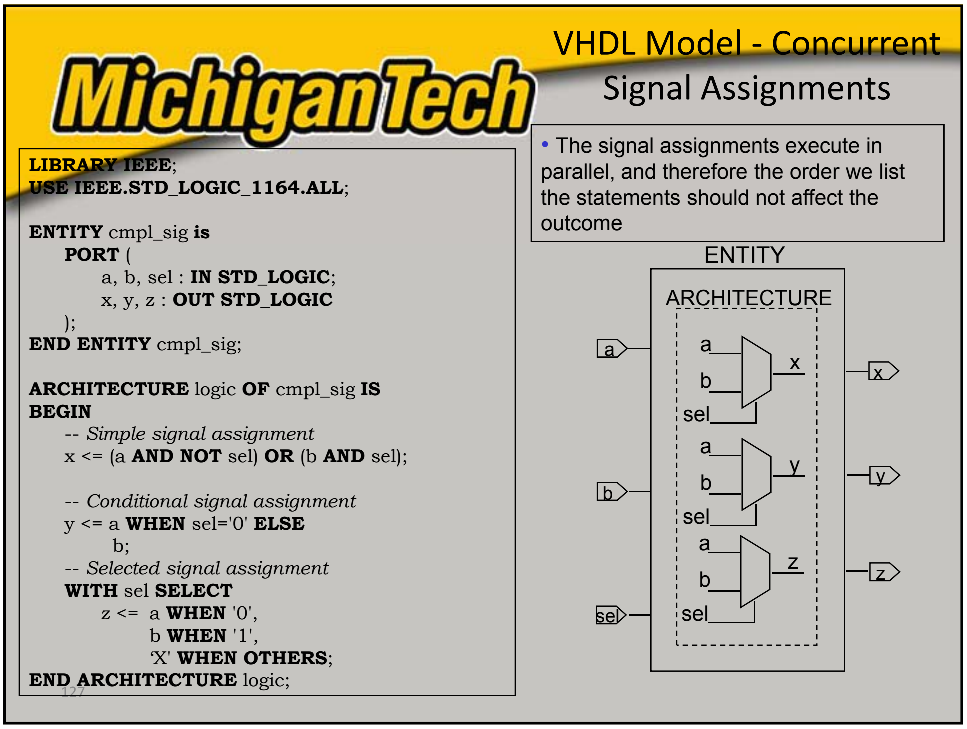
Selected Signal Assignment

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cmpl_sig IS
    PORT (
        a, b, sel : IN STD_LOGIC;
        z : OUT STD_LOGIC
    );
END ENTITY cmpl_sig;

ARCHITECTURE logic OF cmpl_sig IS
BEGIN
      -- Selected signal assignment
    WITH sel SELECT
        z <=  a WHEN '0',
              b WHEN '1',
              '0' WHEN OTHERS;
END ARCHITECTURE logic;
```

sel is of **STD_LOGIC** data type

- *What are the values for a* **STD_LOGIC** *data type*
- *Answer:* **{'0','1','X','Z'…}**
- *Therefore, is the* **WHEN OTHERS** *clause necessary?*
- *Answer:* **YES**

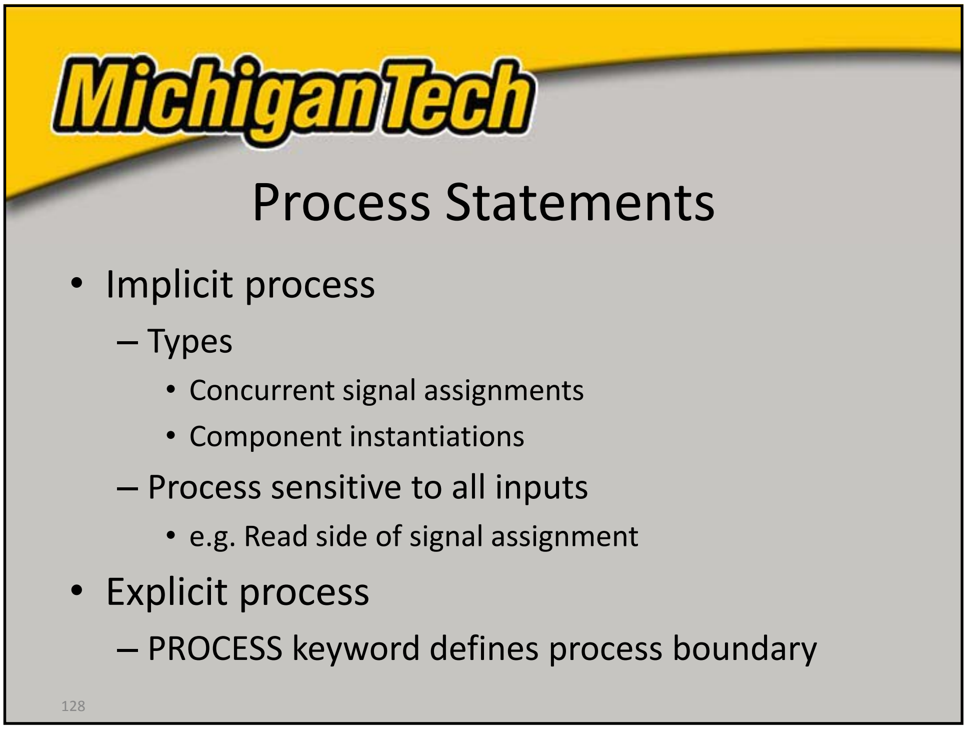
126



VHDL Model - Concurrent Signal Assignments

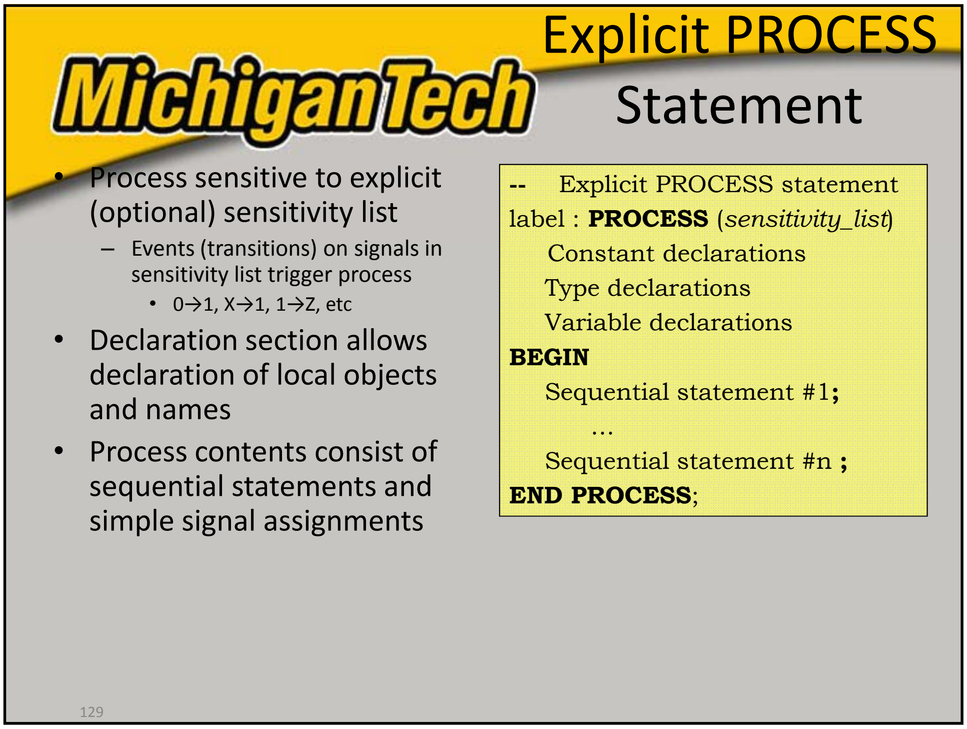
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cmpl_sig is
    PORT (
        a, b, sel : IN STD_LOGIC;
        x, y, z : OUT STD_LOGIC
    );
END ENTITY cmpl_sig;

ARCHITECTURE logic OF cmpl_sig IS
BEGIN
    -- Simple signal assignment
    x <= (a AND NOT sel) OR (b AND sel);

    -- Conditional signal assignment
    y <= a WHEN sel='0' ELSE
          b;
    -- Selected signal assignment
    WITH sel SELECT
        z <=  a WHEN '0',
              b WHEN '1',
             'X' WHEN OTHERS;
END ARCHITECTURE logic;
```

- The signal assignments execute in parallel, and therefore the order we list the statements should not affect the outcome



Process Statements

- Implicit process
  - Types
    - Concurrent signal assignments
    - Component instantiations
  - Process sensitive to all inputs
    - e.g. Read side of signal assignment
- Explicit process
  - PROCESS keyword defines process boundary

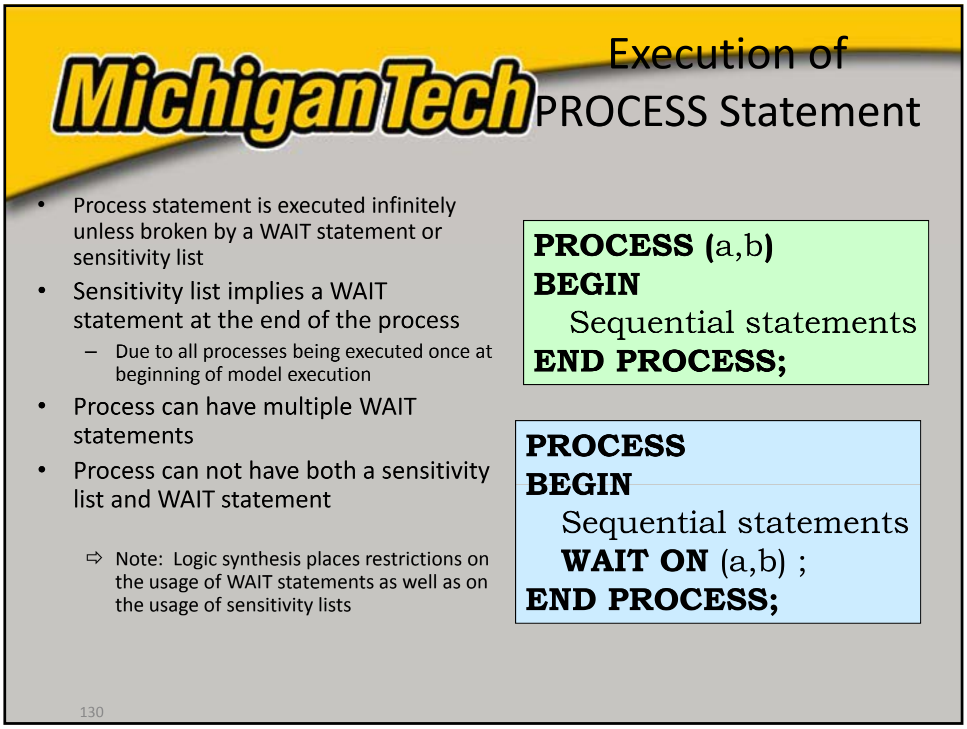


Explicit PROCESS Statement

- Process sensitive to explicit (optional) sensitivity list
  - Events (transitions) on signals in sensitivity list trigger process
    - 0→1, X→1, 1→Z, etc
- Declaration section allows declaration of local objects and names
- Process contents consist of sequential statements and simple signal assignments

```
--    Explicit PROCESS statement
label : PROCESS (sensitivity_list)
      Constant declarations
      Type declarations
      Variable declarations
BEGIN
      Sequential statement #1;
            …
      Sequential statement #n ;
END PROCESS;
```

129



Execution of PROCESS Statement

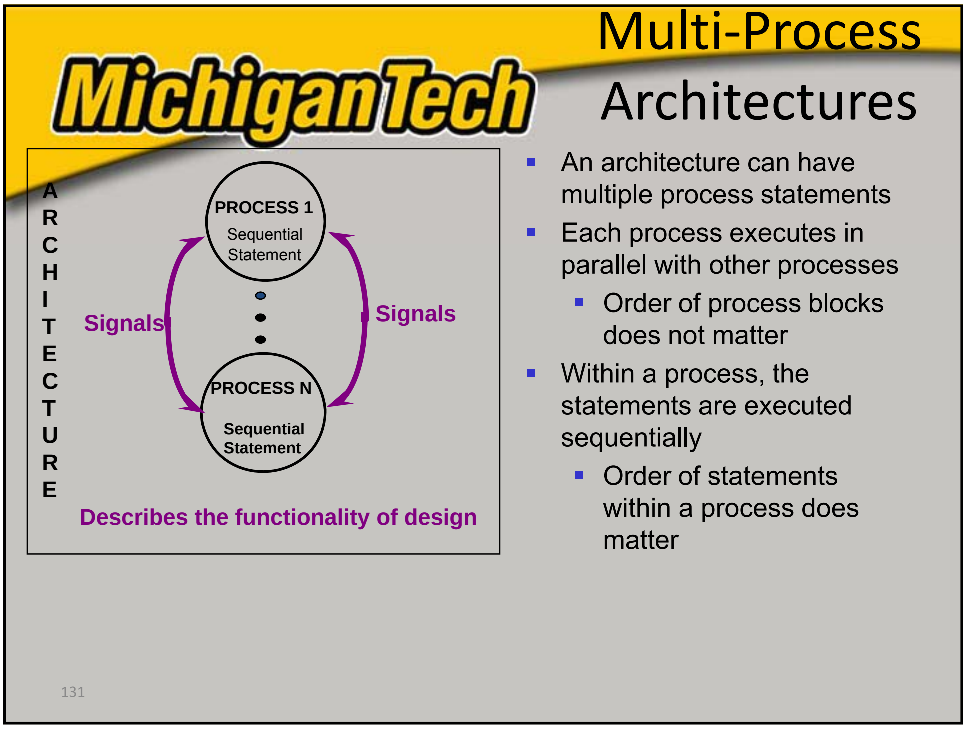
- Process statement is executed infinitely unless broken by a WAIT statement or sensitivity list
- Sensitivity list implies a WAIT statement at the end of the process
  - Due to all processes being executed once at beginning of model execution
- Process can have multiple WAIT statements
- Process can not have both a sensitivity list and WAIT statement

⇨ Note: Logic synthesis places restrictions on the usage of WAIT statements as well as on the usage of sensitivity lists

```
PROCESS (a,b)
BEGIN
   Sequential statements
END PROCESS;
```

```
PROCESS
BEGIN
   Sequential statements
   WAIT ON (a,b) ;
END PROCESS;
```

130

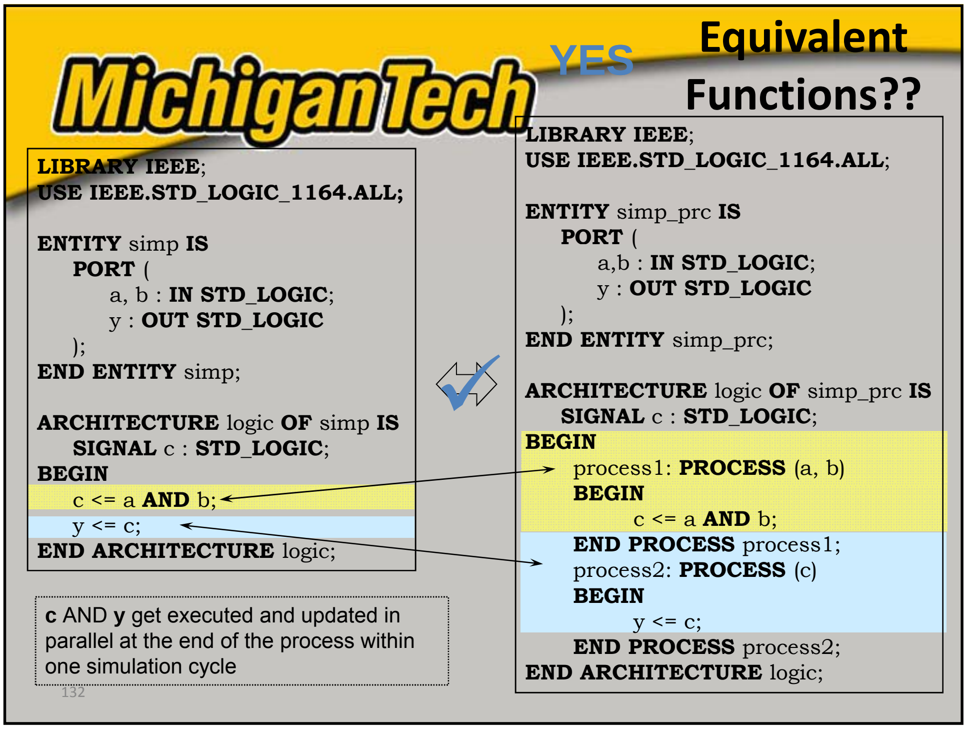


Multi-Process Architectures

Describes the functionality of design

- An architecture can have multiple process statements
- Each process executes in parallel with other processes
  - Order of process blocks does not matter
- Within a process, the statements are executed sequentially
  - Order of statements within a process does matter

131



Equivalent Functions??

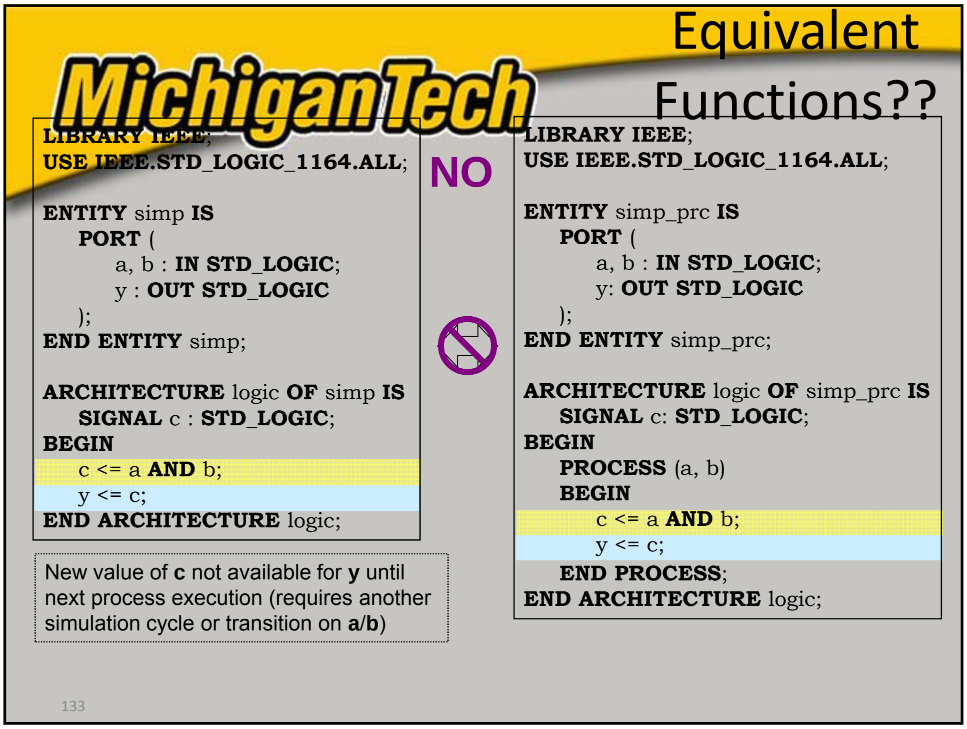
YES

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp IS
    PORT (
        a, b : IN STD_LOGIC;
        y : OUT STD_LOGIC
    );
END ENTITY simp;

ARCHITECTURE logic OF simp IS
    SIGNAL c : STD_LOGIC;
BEGIN
    c <= a AND b;
    y <= c;
END ARCHITECTURE logic;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp_prc IS
    PORT (
        a,b : IN STD_LOGIC;
        y : OUT STD_LOGIC
    );
END ENTITY simp_prc;

ARCHITECTURE logic OF simp_prc IS
    SIGNAL c : STD_LOGIC;
BEGIN
    process1: PROCESS (a, b)
    BEGIN
        c <= a AND b;
    END PROCESS process1;
    process2: PROCESS (c)
    BEGIN
        y <= c;
    END PROCESS process2;
END ARCHITECTURE logic;
```

c AND **y** get executed and updated in parallel at the end of the process within one simulation cycle

132



Equivalent Functions??

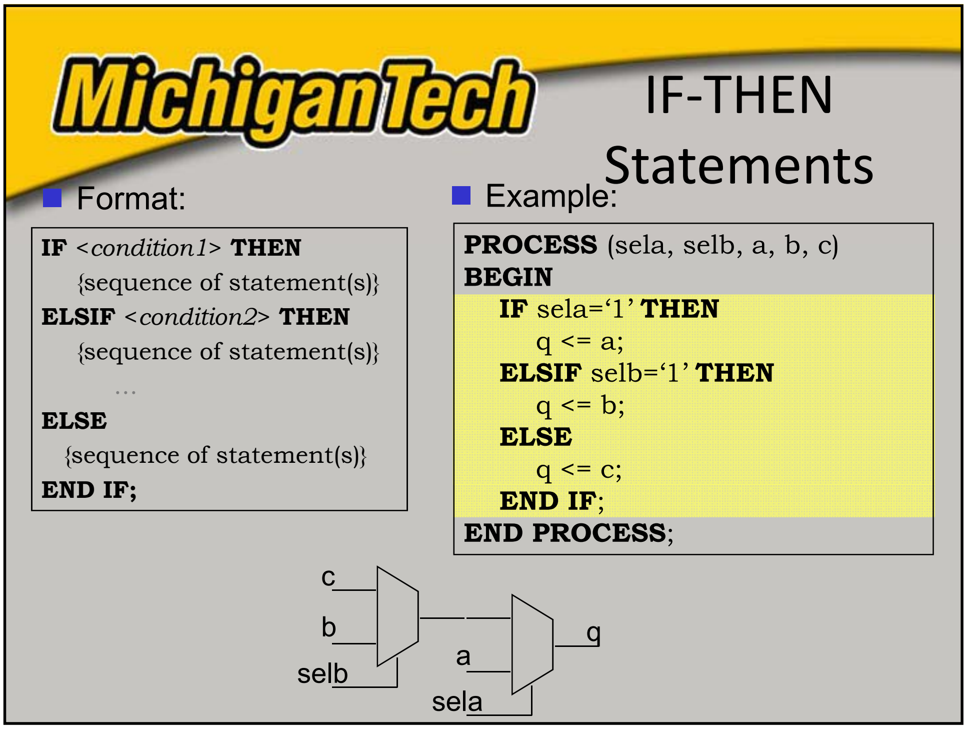
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp IS
    PORT (
        a, b : IN STD_LOGIC;
        y : OUT STD_LOGIC
    );
END ENTITY simp;

ARCHITECTURE logic OF simp IS
    SIGNAL c : STD_LOGIC;
BEGIN
    c <= a AND b;
    y <= c;
END ARCHITECTURE logic;
```

NO

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp_prc IS
    PORT (
        a, b : IN STD_LOGIC;
        y: OUT STD_LOGIC
    );
END ENTITY simp_prc;

ARCHITECTURE logic OF simp_prc IS
    SIGNAL c: STD_LOGIC;
BEGIN
    PROCESS (a, b)
    BEGIN
        c <= a AND b;
        y <= c;
    END PROCESS;
END ARCHITECTURE logic;
```

New value of **c** not available for **y** until next process execution (requires another simulation cycle or transition on **a**/**b**)

133



IF-THEN Statements

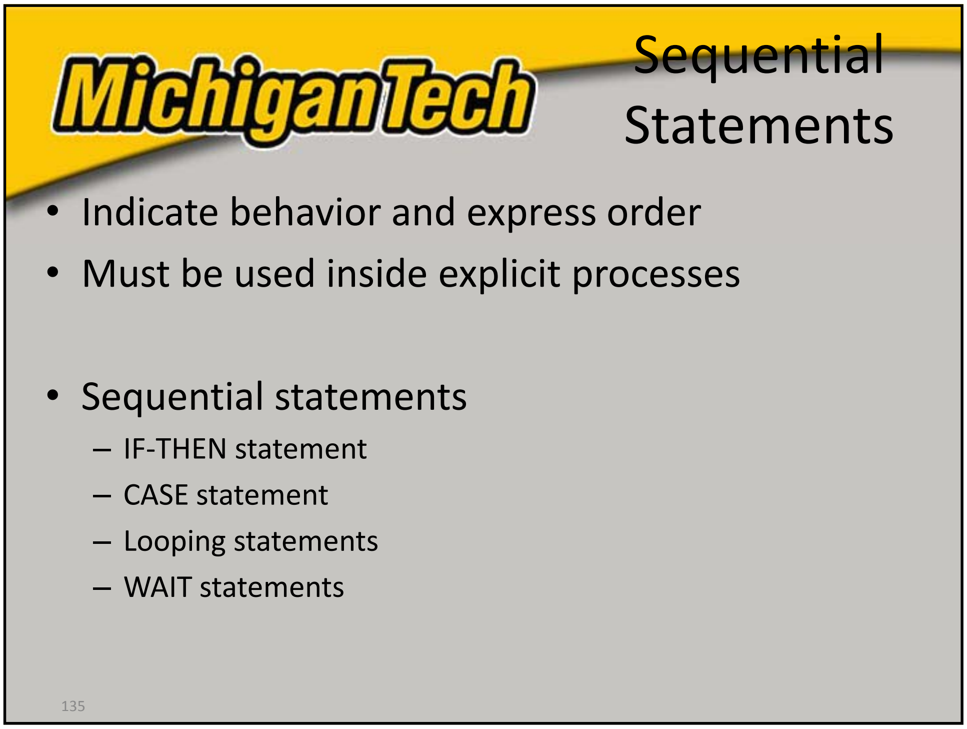
- Format:

```
IF <condition1> THEN
    {sequence of statement(s)}
ELSIF <condition2> THEN
    {sequence of statement(s)}
        …
ELSE
    {sequence of statement(s)}
END IF;
```

- Example:

```
PROCESS (sela, selb, a, b, c)
BEGIN
    IF sela='1' THEN
        q <= a;
    ELSIF selb='1' THEN
        q <= b;
    ELSE
        q <= c;
    END IF;
END PROCESS;
```

c
b
selb
a
sela
q

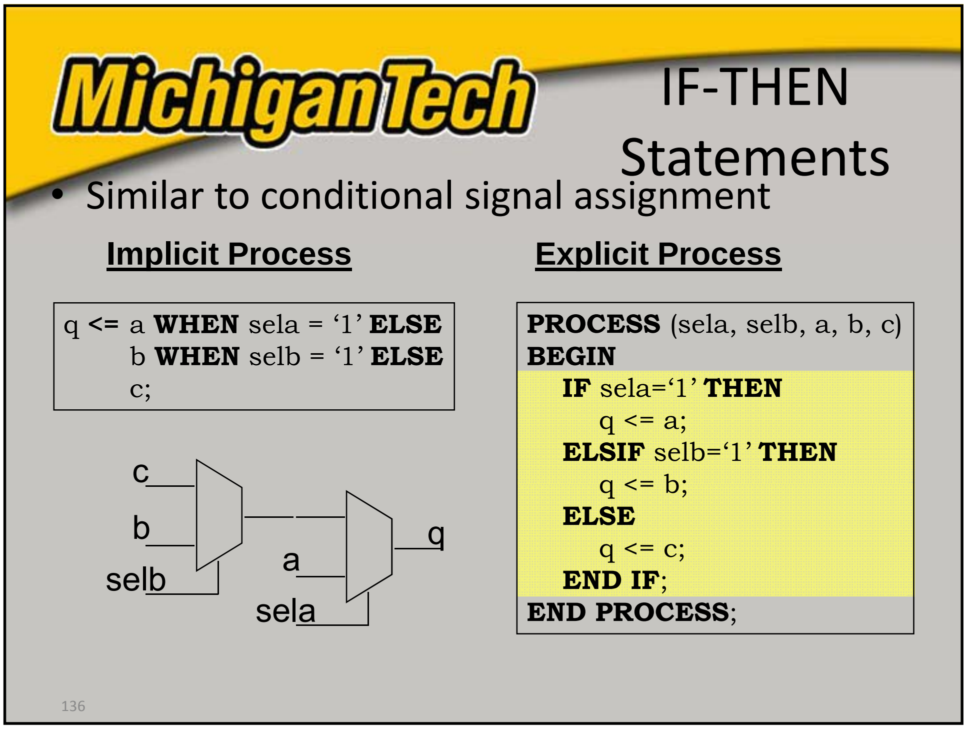


Sequential Statements

- Indicate behavior and express order
- Must be used inside explicit processes

- Sequential statements
  - IF-THEN statement
  - CASE statement
  - Looping statements
  - WAIT statements

135



IF-THEN Statements

- Similar to conditional signal assignment

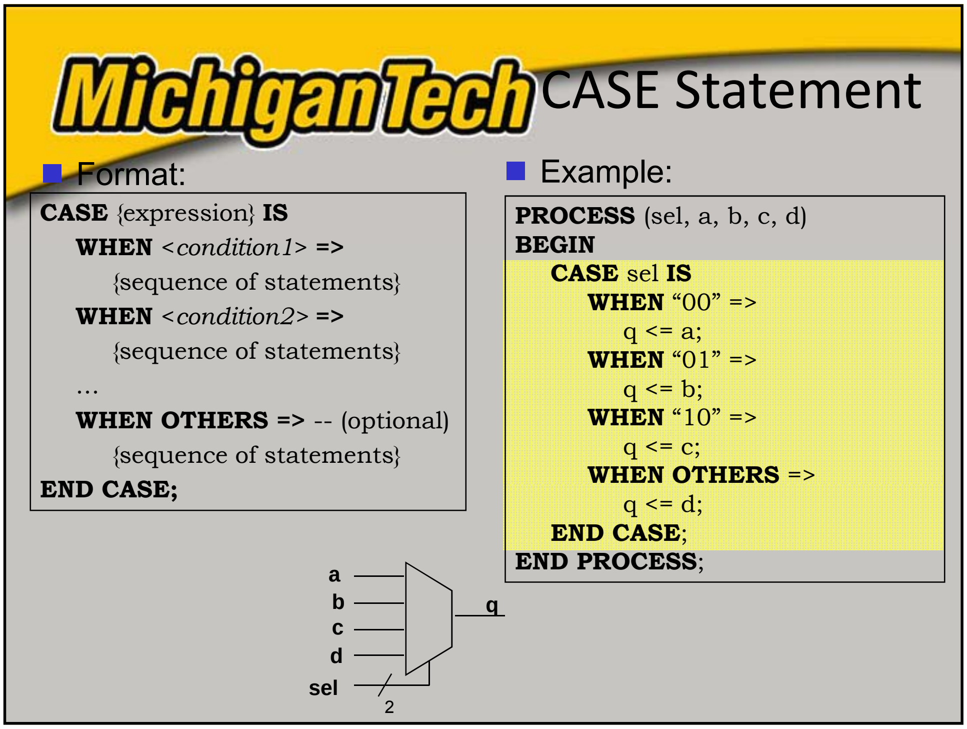
Implicit Process

```
q <= a WHEN sela = '1' ELSE
     b WHEN selb = '1' ELSE
     c;
```

Explicit Process

```
PROCESS (sela, selb, a, b, c)
BEGIN
   IF sela='1' THEN
      q <= a;
   ELSIF selb='1' THEN
      q <= b;
   ELSE
      q <= c;
   END IF;
END PROCESS;
```

136



MichiganTech

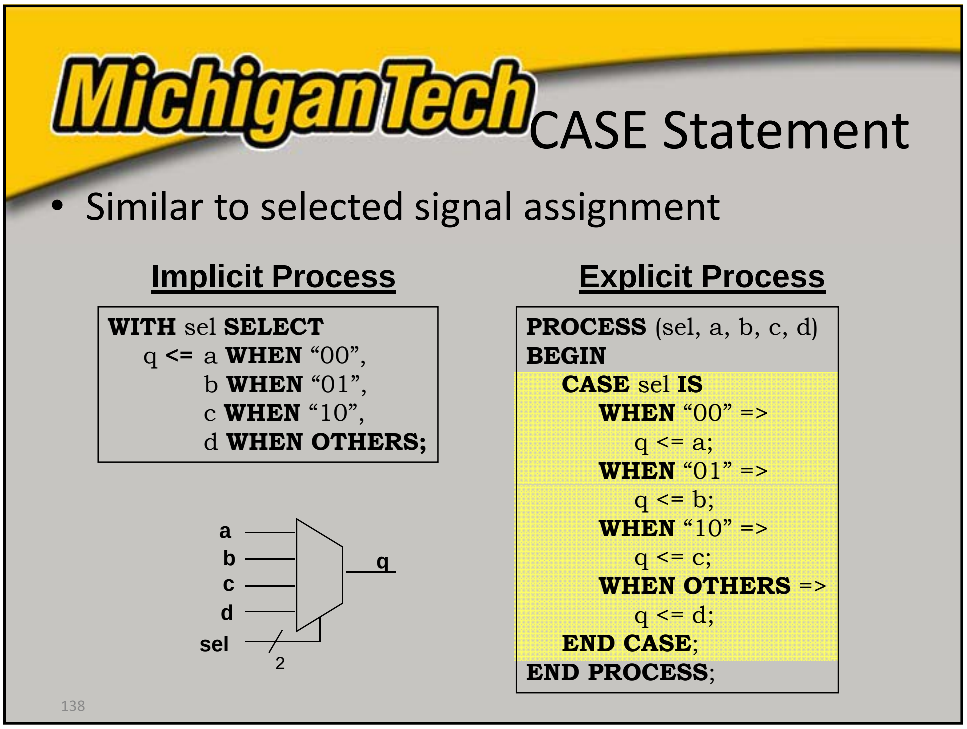
CASE Statement

- Format:

```
CASE {expression} IS
   WHEN <condition1> =>
      {sequence of statements}
   WHEN <condition2> =>
      {sequence of statements}
   …
   WHEN OTHERS => -- (optional)
      {sequence of statements}
END CASE;
```

- Example:

```
PROCESS (sel, a, b, c, d)
BEGIN
   CASE sel IS
      WHEN “00” =>
         q <= a;
      WHEN “01” =>
         q <= b;
      WHEN “10” =>
         q <= c;
      WHEN OTHERS =>
         q <= d;
   END CASE;
END PROCESS;
```



MichiganTech

CASE Statement

- Similar to selected signal assignment

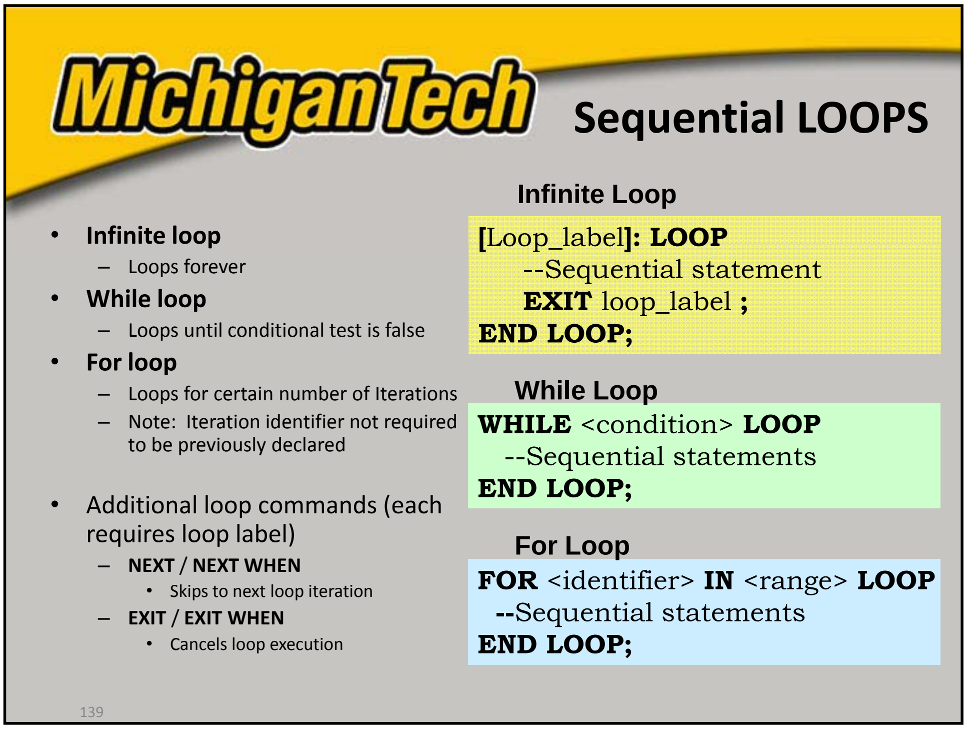
Implicit Process

```
WITH sel SELECT
   q <= a WHEN “00”,
        b WHEN “01”,
        c WHEN “10”,
        d WHEN OTHERS;
```

Explicit Process

```
PROCESS (sel, a, b, c, d)
BEGIN
   CASE sel IS
      WHEN “00” =>
         q <= a;
      WHEN “01” =>
         q <= b;
      WHEN “10” =>
         q <= c;
      WHEN OTHERS =>
         q <= d;
   END CASE;
END PROCESS;
```

138



MichiganTech

Sequential LOOPS

- **Infinite loop**
  - Loops forever
- **While loop**
  - Loops until conditional test is false
- **For loop**
  - Loops for certain number of Iterations
  - Note: Iteration identifier not required to be previously declared
- Additional loop commands (each requires loop label)
  - **NEXT / NEXT WHEN**
    - Skips to next loop iteration
  - **EXIT / EXIT WHEN**
    - Cancels loop execution

Infinite Loop

```
[Loop_label]: LOOP
     --Sequential statement
     EXIT loop_label ;
END LOOP;
```

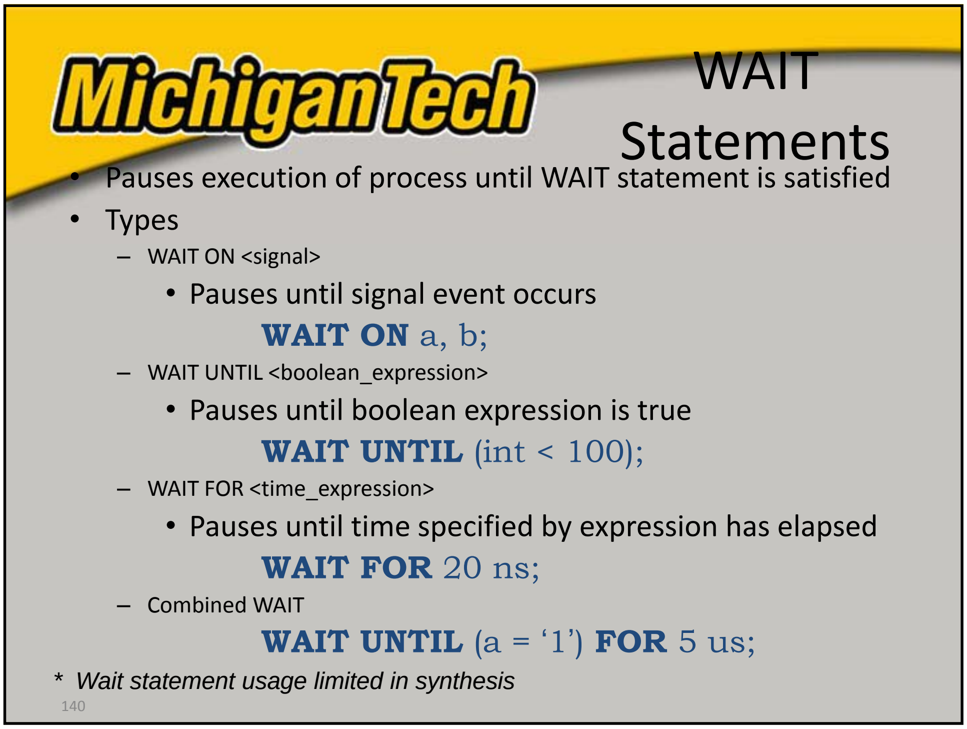
While Loop

```
WHILE <condition> LOOP
   --Sequential statements
END LOOP;
```

For Loop

```
FOR <identifier> IN <range> LOOP
  --Sequential statements
END LOOP;
```

139



MichiganTech

WAIT Statements

- Pauses execution of process until WAIT statement is satisfied
- Types
  - WAIT ON <signal>
    - Pauses until signal event occurs

      `WAIT ON a, b;`
  - WAIT UNTIL <boolean_expression>
    - Pauses until boolean expression is true

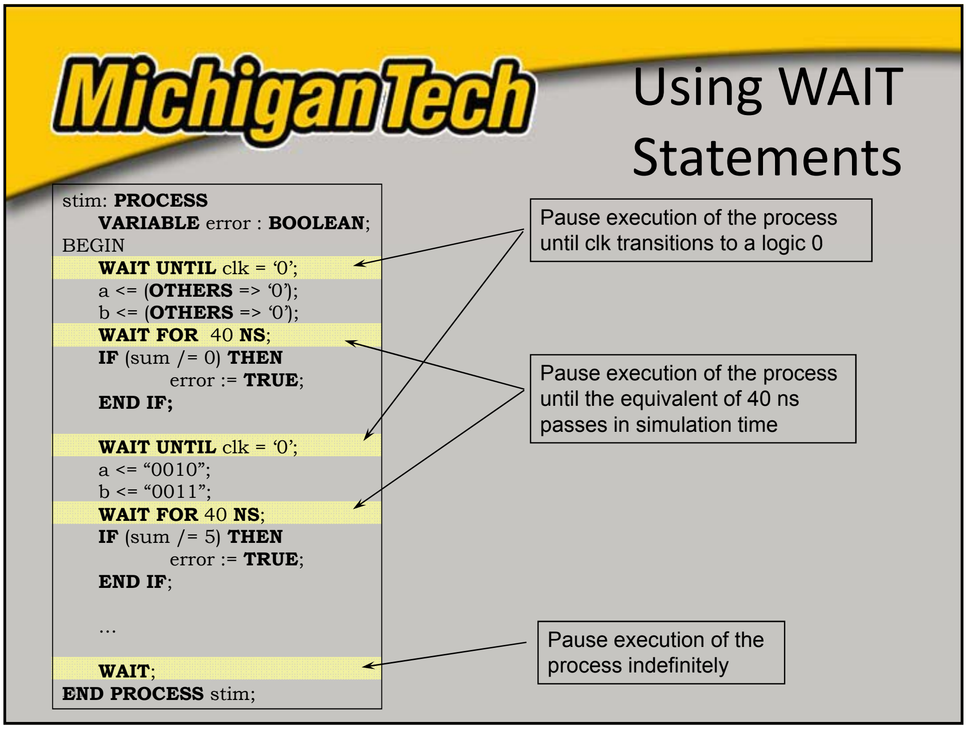
      `WAIT UNTIL (int < 100);`
  - WAIT FOR <time_expression>
    - Pauses until time specified by expression has elapsed

      `WAIT FOR 20 ns;`
  - Combined WAIT

    `WAIT UNTIL (a = '1') FOR 5 us;`

* *Wait statement usage limited in synthesis*

140



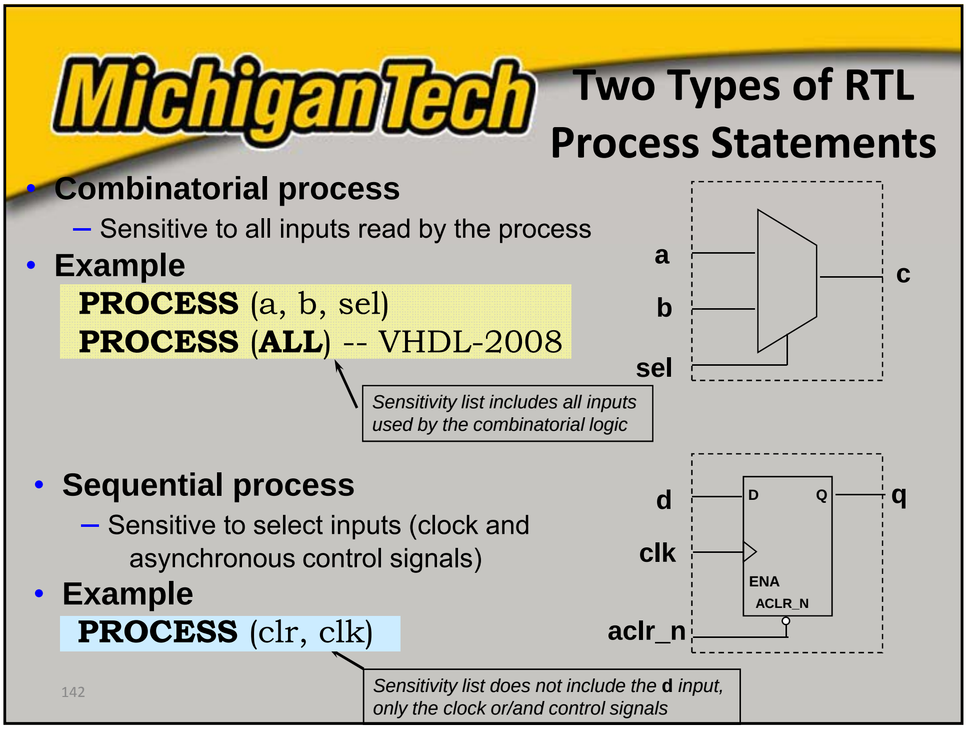
Using WAIT Statements

```
stim: PROCESS
     VARIABLE error : BOOLEAN;
BEGIN
     WAIT UNTIL clk = '0';
     a <= (OTHERS => '0');
     b <= (OTHERS => '0');
     WAIT FOR  40 NS;
     IF (sum /= 0) THEN
               error := TRUE;
     END IF;

     WAIT UNTIL clk = '0';
     a <= "0010";
     b <= "0011";
     WAIT FOR 40 NS;
     IF (sum /= 5) THEN
               error := TRUE;
     END IF;

     ...

     WAIT;
END PROCESS stim;
```

Pause execution of the process until clk transitions to a logic 0

Pause execution of the process until the equivalent of 40 ns passes in simulation time

Pause execution of the process indefinitely



Two Types of RTL Process Statements

- Combinatorial process
  - Sensitive to all inputs read by the process
- Example

PROCESS (a, b, sel)
PROCESS (**ALL**) -- VHDL-2008

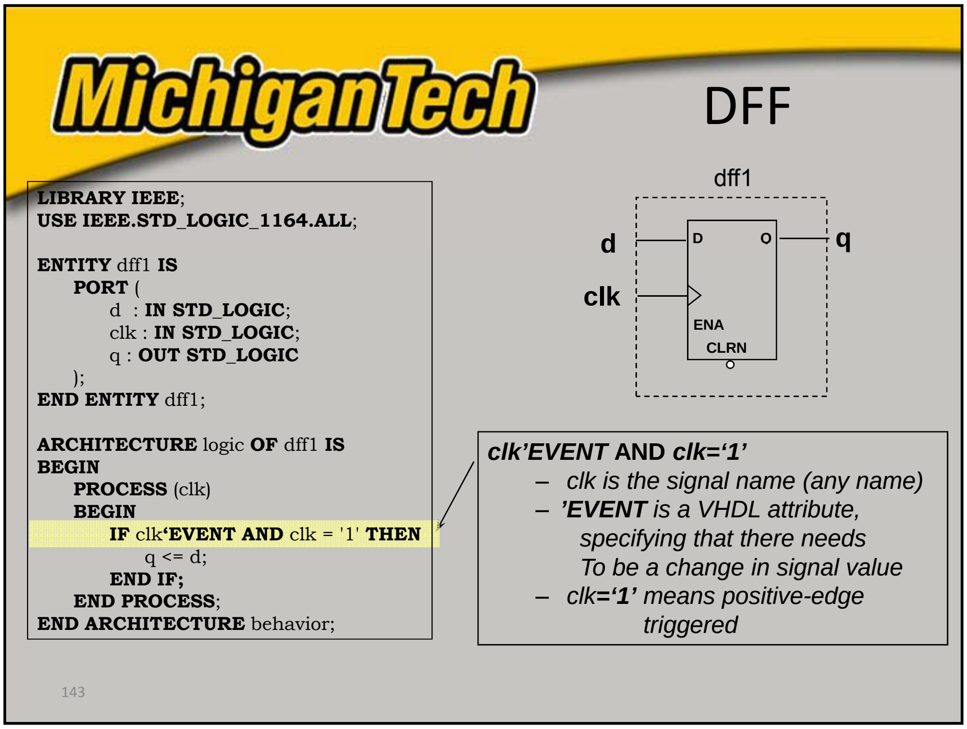
Sensitivity list includes all inputs used by the combinatorial logic

- Sequential process
  - Sensitive to select inputs (clock and asynchronous control signals)
- Example

PROCESS (clr, clk)

*Sensitivity list does not include the **d** input, only the clock or/and control signals*

142



DFF

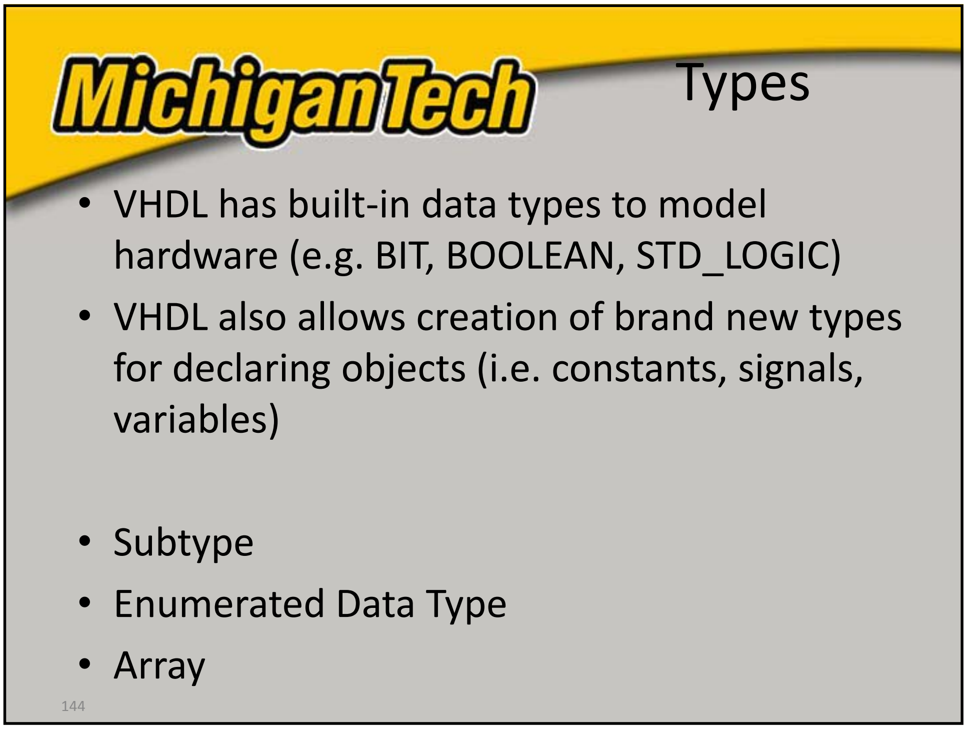
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY dff1 IS
     PORT (
          d  : IN STD_LOGIC;
          clk : IN STD_LOGIC;
          q : OUT STD_LOGIC
     );
END ENTITY dff1;

ARCHITECTURE logic OF dff1 IS
BEGIN
     PROCESS (clk)
     BEGIN
          IF clk'EVENT AND clk = '1' THEN
               q <= d;
          END IF;
     END PROCESS;
END ARCHITECTURE behavior;
```

dff1
d
clk
q
D
O
ENA
CLRN

clk'EVENT* AND *clk='1'

- *clk is the signal name (any name)*
- ***'EVENT** is a VHDL attribute, specifying that there needs To be a change in signal value*
- ***clk='1'** means positive-edge triggered*

143

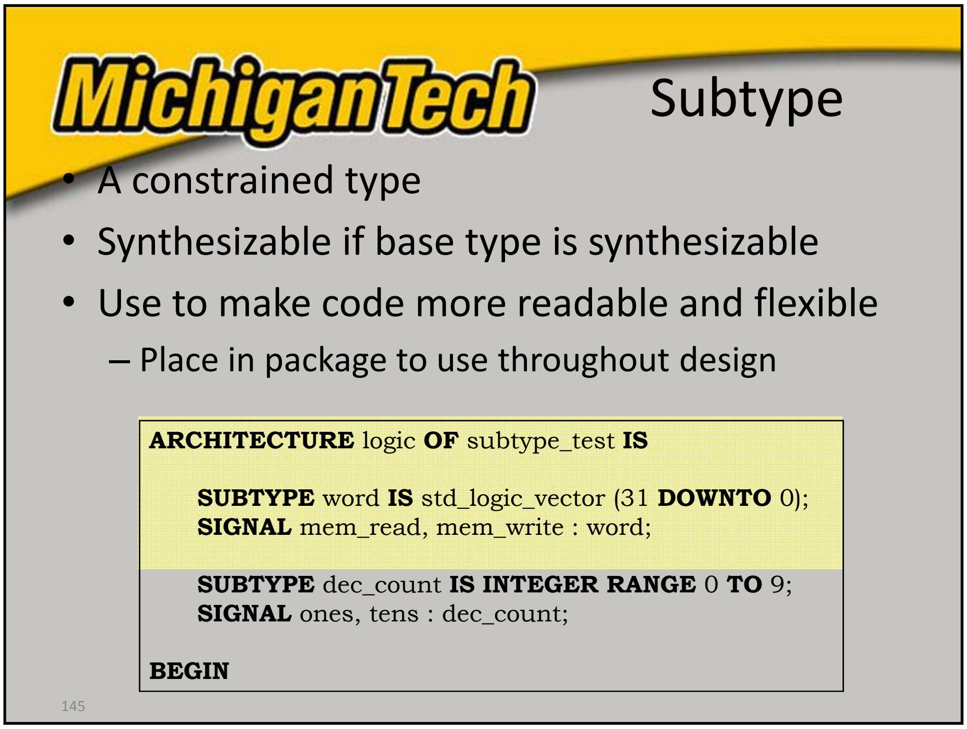


Types

- VHDL has built-in data types to model hardware (e.g. BIT, BOOLEAN, STD_LOGIC)
- VHDL also allows creation of brand new types for declaring objects (i.e. constants, signals, variables)

- Subtype
- Enumerated Data Type
- Array

144

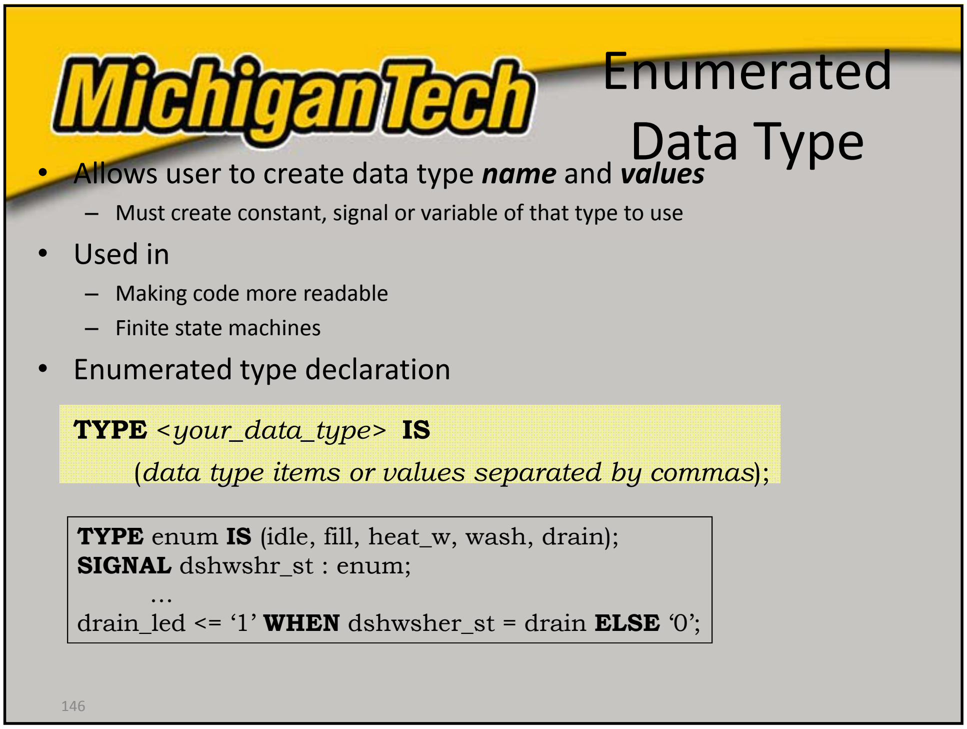


Subtype

- A constrained type
- Synthesizable if base type is synthesizable
- Use to make code more readable and flexible
  - Place in package to use throughout design

```
ARCHITECTURE logic OF subtype_test IS

    SUBTYPE word IS std_logic_vector (31 DOWNTO 0);
    SIGNAL mem_read, mem_write : word;

    SUBTYPE dec_count IS INTEGER RANGE 0 TO 9;
    SIGNAL ones, tens : dec_count;

BEGIN
```

145



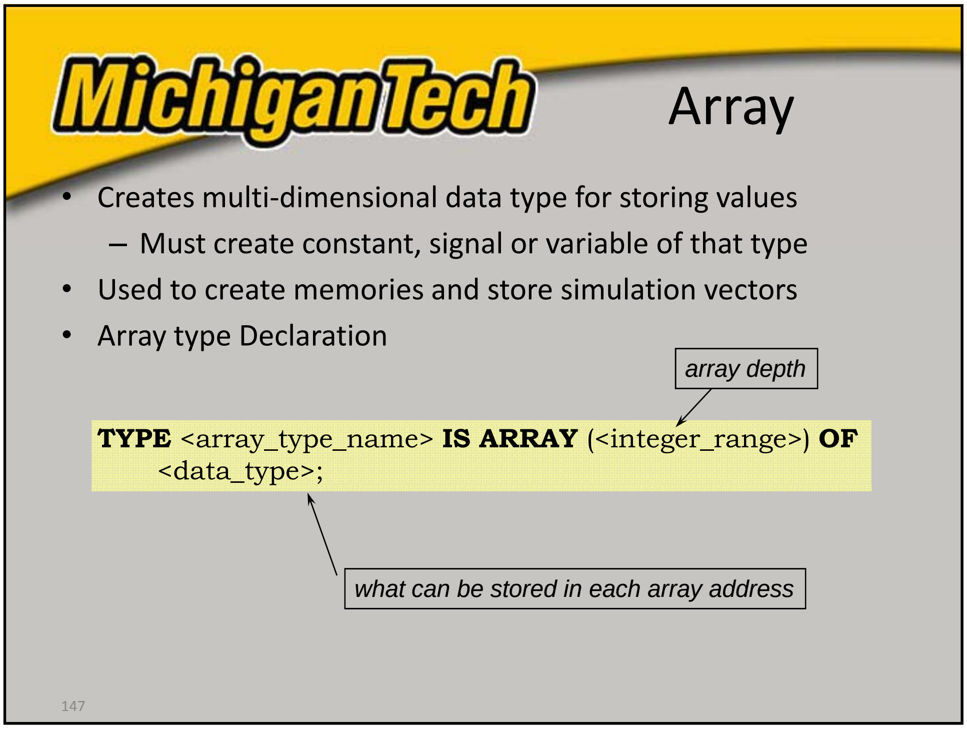
Enumerated Data Type

- Allows user to create data type ***name*** and ***values***
  - Must create constant, signal or variable of that type to use
- Used in
  - Making code more readable
  - Finite state machines
- Enumerated type declaration

```
TYPE <your_data_type>  IS
        (data type items or values separated by commas);
```

```
TYPE enum IS (idle, fill, heat_w, wash, drain);
SIGNAL dshwshr_st : enum;
        …
drain_led <= ‘1’ WHEN dshwsher_st = drain ELSE ‘0’;
```

146



Array

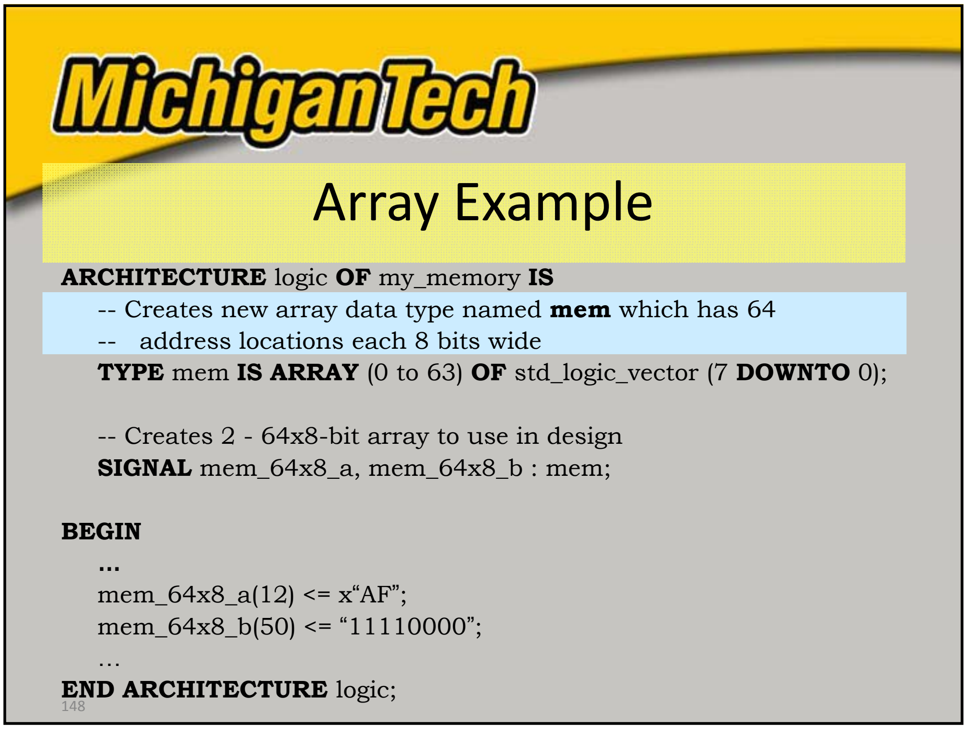
- Creates multi-dimensional data type for storing values
  - Must create constant, signal or variable of that type
- Used to create memories and store simulation vectors
- Array type Declaration

array depth

```
TYPE <array_type_name> IS ARRAY (<integer_range>) OF
      <data_type>;
```

what can be stored in each array address

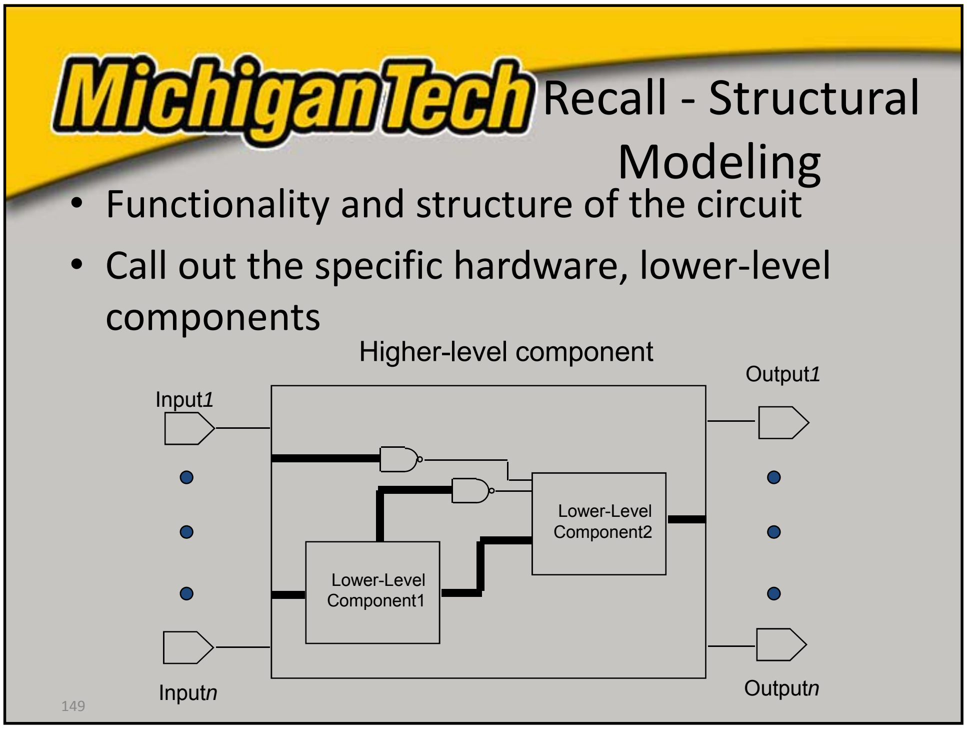
147



Array Example

```
ARCHITECTURE logic OF my_memory IS
    -- Creates new array data type named mem which has 64
    --   address locations each 8 bits wide
    TYPE mem IS ARRAY (0 to 63) OF std_logic_vector (7 DOWNTO 0);

    -- Creates 2 - 64x8-bit array to use in design
    SIGNAL mem_64x8_a, mem_64x8_b : mem;

BEGIN
    …
    mem_64x8_a(12) <= x“AF”;
    mem_64x8_b(50) <= “11110000”;
    …
END ARCHITECTURE logic;
```

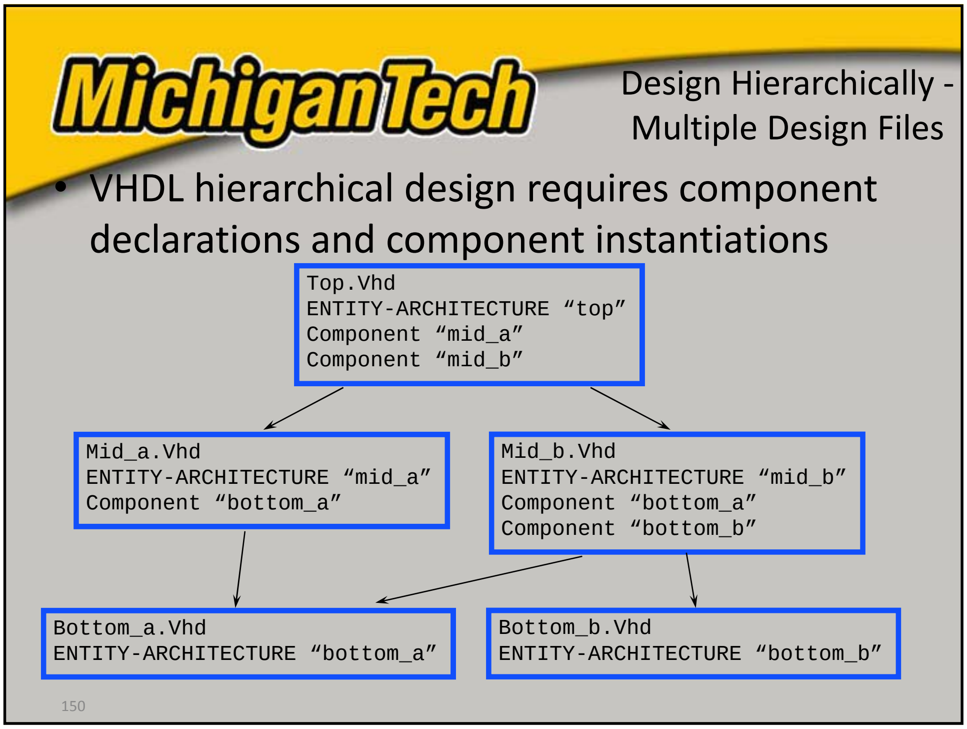
148



Recall - Structural Modeling

- Functionality and structure of the circuit
- Call out the specific hardware, lower-level components

149



Design Hierarchically - Multiple Design Files

- VHDL hierarchical design requires component declarations and component instantiations

```
Top.Vhd
ENTITY-ARCHITECTURE "top"
Component "mid_a"
Component "mid_b"
```

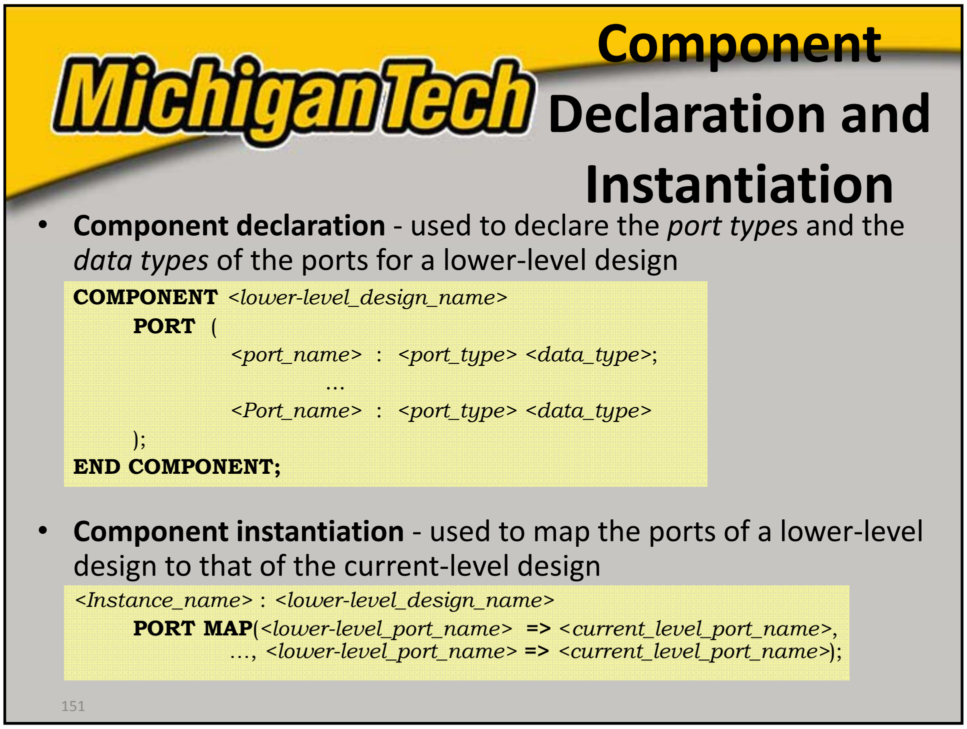
```
Mid_a.Vhd
ENTITY-ARCHITECTURE "mid_a"
Component "bottom_a"
```

```
Mid_b.Vhd
ENTITY-ARCHITECTURE "mid_b"
Component "bottom_a"
Component "bottom_b"
```

```
Bottom_a.Vhd
ENTITY-ARCHITECTURE "bottom_a"
```

```
Bottom_b.Vhd
ENTITY-ARCHITECTURE "bottom_b"
```

150



Component Declaration and Instantiation

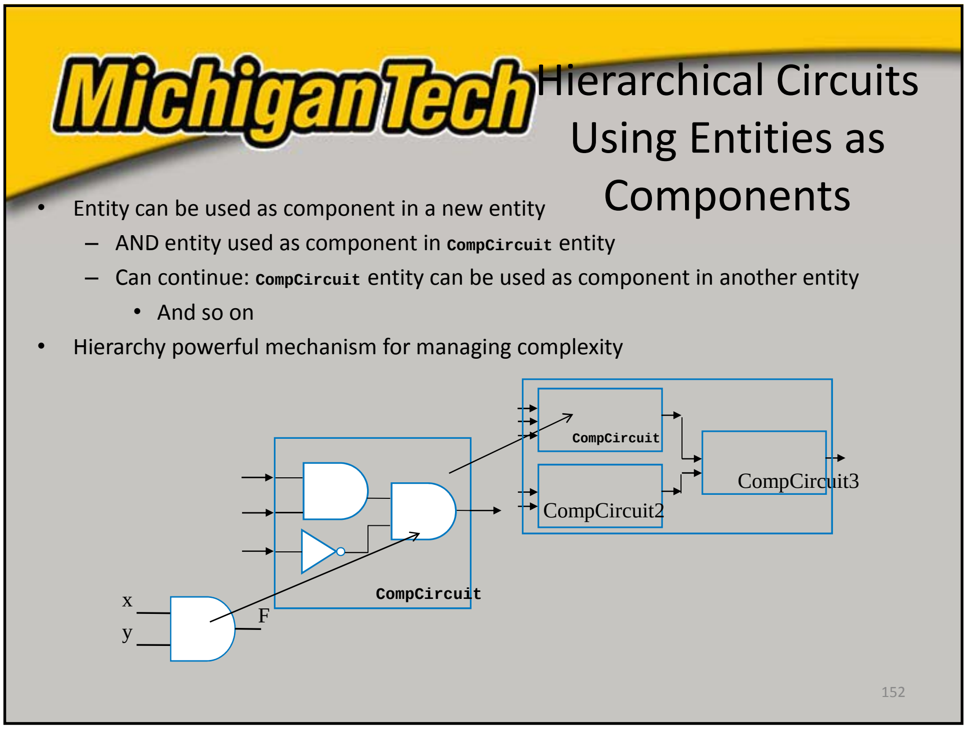
- **Component declaration** - used to declare the *port types* and the *data types* of the ports for a lower-level design

```
COMPONENT <lower-level_design_name>
      PORT  (
                <port_name>  :  <port_type> <data_type>;
                     …
                <Port_name>  :  <port_type> <data_type>
      );
END COMPONENT;
```

- **Component instantiation** - used to map the ports of a lower-level design to that of the current-level design

```
<Instance_name> : <lower-level_design_name>
      PORT MAP(<lower-level_port_name>  => <current_level_port_name>,
                …, <lower-level_port_name> => <current_level_port_name>);
```

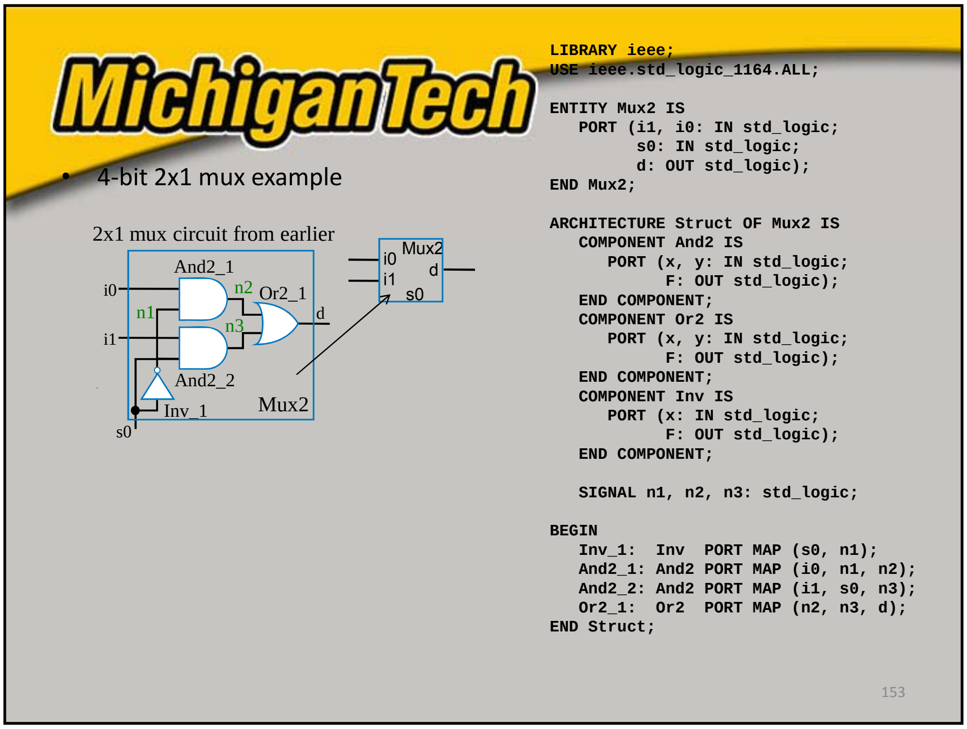
151



Hierarchical Circuits Using Entities as Components

- Entity can be used as component in a new entity
  - AND entity used as component in `CompCircuit` entity
  - Can continue: `CompCircuit` entity can be used as component in another entity
    - And so on
- Hierarchy powerful mechanism for managing complexity

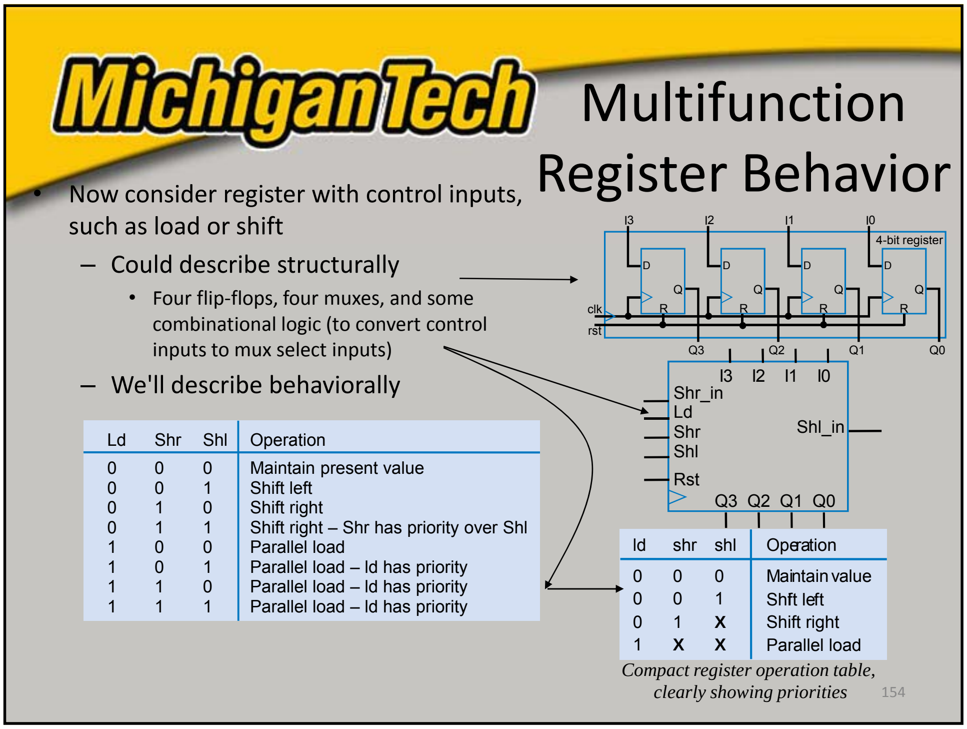
152



- 4-bit 2x1 mux example

2x1 mux circuit from earlier

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Mux2 IS
   PORT (i1, i0: IN std_logic;
         s0: IN std_logic;
         d: OUT std_logic);
END Mux2;

ARCHITECTURE Struct OF Mux2 IS
   COMPONENT And2 IS
      PORT (x, y: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;
   COMPONENT Or2 IS
      PORT (x, y: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;
   COMPONENT Inv IS
      PORT (x: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;

   SIGNAL n1, n2, n3: std_logic;

BEGIN
   Inv_1:  Inv  PORT MAP (s0, n1);
   And2_1: And2 PORT MAP (i0, n1, n2);
   And2_2: And2 PORT MAP (i1, s0, n3);
   Or2_1:  Or2  PORT MAP (n2, n3, d);
END Struct;
```



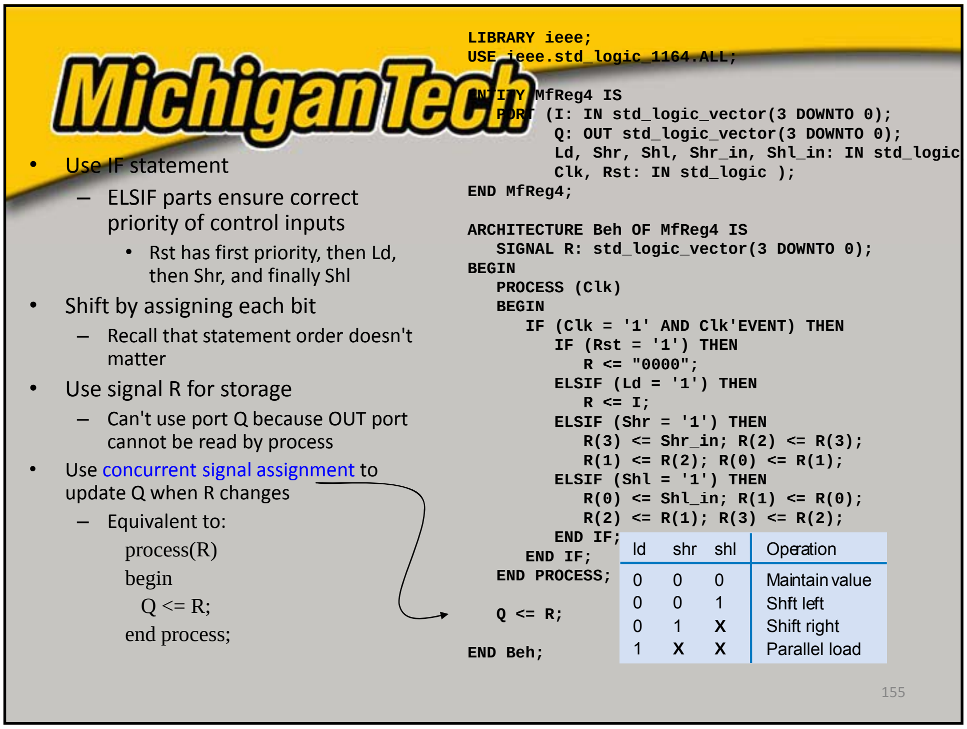
Multifunction Register Behavior

- Now consider register with control inputs, such as load or shift
  - Could describe structurally
    - Four flip-flops, four muxes, and some combinational logic (to convert control inputs to mux select inputs)
  - We'll describe behaviorally

Ld	Shr	Shl	Operation
0	0	0	Maintain present value
0	0	1	Shift left
0	1	0	Shift right
0	1	1	Shift right – Shr has priority over Shl
1	0	0	Parallel load
1	0	1	Parallel load – ld has priority
1	1	0	Parallel load – ld has priority
1	1	1	Parallel load – ld has priority

ld	shr	shl	Operation
0	0	0	Maintain value
0	0	1	Shft left
0	1	X	Shift right
1	X	X	Parallel load

Compact register operation table, clearly showing priorities



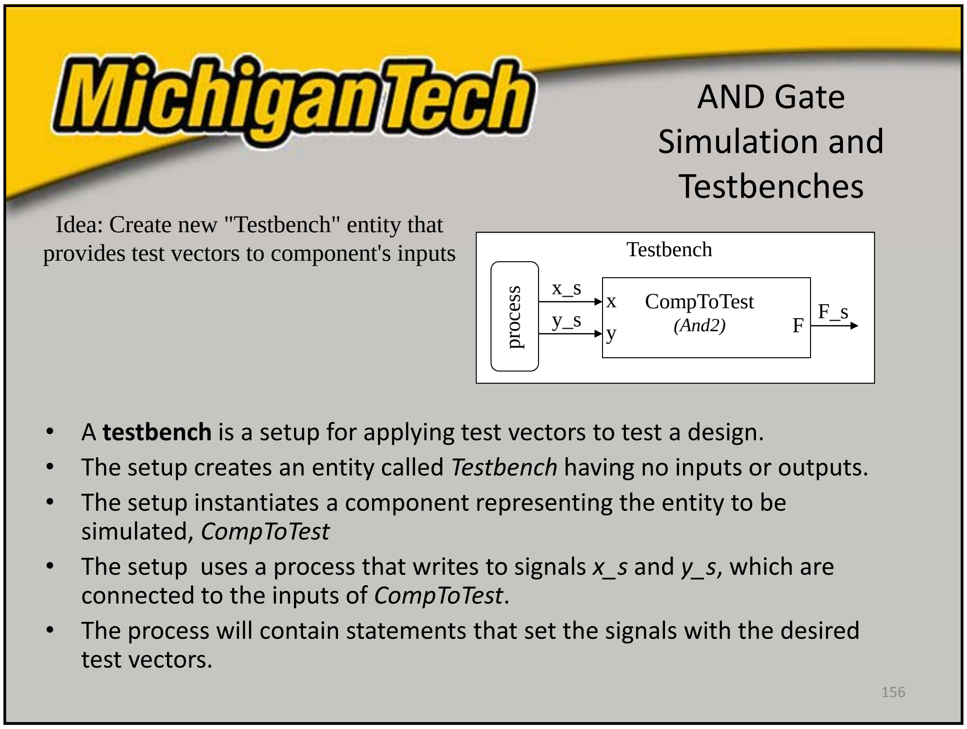
- Use IF statement
  - ELSIF parts ensure correct priority of control inputs
    - Rst has first priority, then Ld, then Shr, and finally Shl
- Shift by assigning each bit
  - Recall that statement order doesn't matter
- Use signal R for storage
  - Can't use port Q because OUT port cannot be read by process
- Use concurrent signal assignment to update Q when R changes
  - Equivalent to:

```
process(R)
begin
  Q <= R;
end process;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY MfReg4 IS
   PORT (I: IN std_logic_vector(3 DOWNTO 0);
         Q: OUT std_logic_vector(3 DOWNTO 0);
         Ld, Shr, Shl, Shr_in, Shl_in: IN std_logic
         Clk, Rst: IN std_logic );
END MfReg4;

ARCHITECTURE Beh OF MfReg4 IS
   SIGNAL R: std_logic_vector(3 DOWNTO 0);
BEGIN
   PROCESS (Clk)
   BEGIN
      IF (Clk = '1' AND Clk'EVENT) THEN
         IF (Rst = '1') THEN
            R <= "0000";
         ELSIF (Ld = '1') THEN
            R <= I;
         ELSIF (Shr = '1') THEN
            R(3) <= Shr_in; R(2) <= R(3);
            R(1) <= R(2); R(0) <= R(1);
         ELSIF (Shl = '1') THEN
            R(0) <= Shl_in; R(1) <= R(0);
            R(2) <= R(1); R(3) <= R(2);
         END IF;
      END IF;
   END PROCESS;

   Q <= R;

END Beh;
```

ld	shr	shl	Operation
0	0	0	Maintain value
0	0	1	Shft left
0	1	X	Shift right
1	X	X	Parallel load

155



AND Gate Simulation and Testbenches

Idea: Create new "Testbench" entity that provides test vectors to component's inputs

Testbench

process

x_s

y_s

x

y

CompToTest

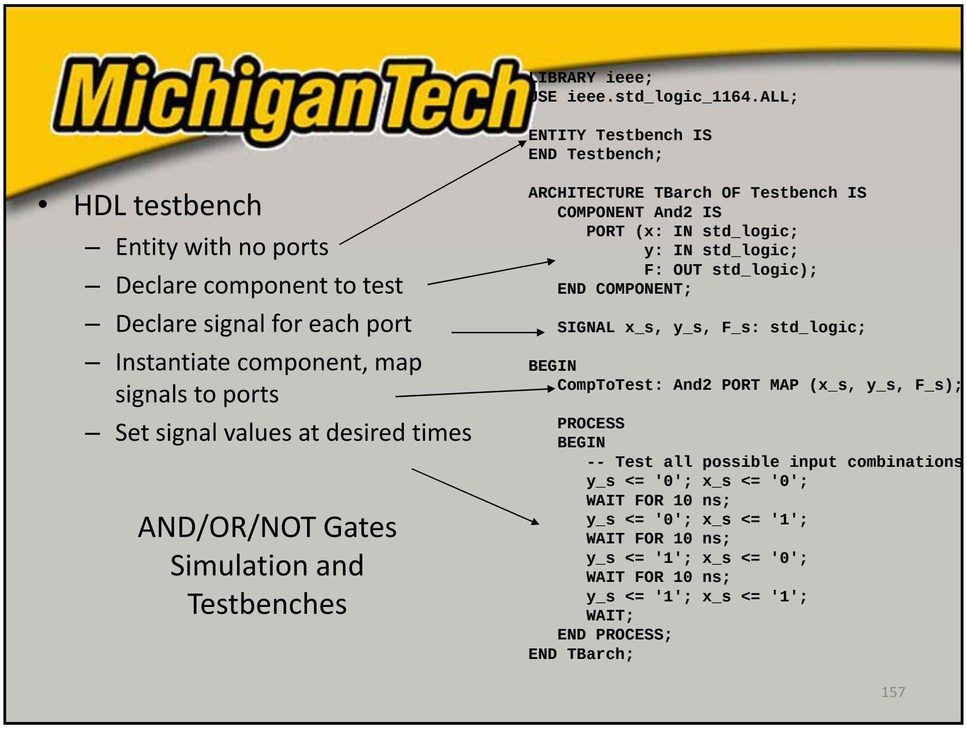
(And2)

F

F_s

- A **testbench** is a setup for applying test vectors to test a design.
- The setup creates an entity called *Testbench* having no inputs or outputs.
- The setup instantiates a component representing the entity to be simulated, *CompToTest*
- The setup uses a process that writes to signals *x_s* and *y_s*, which are connected to the inputs of *CompToTest*.
- The process will contain statements that set the signals with the desired test vectors.

156

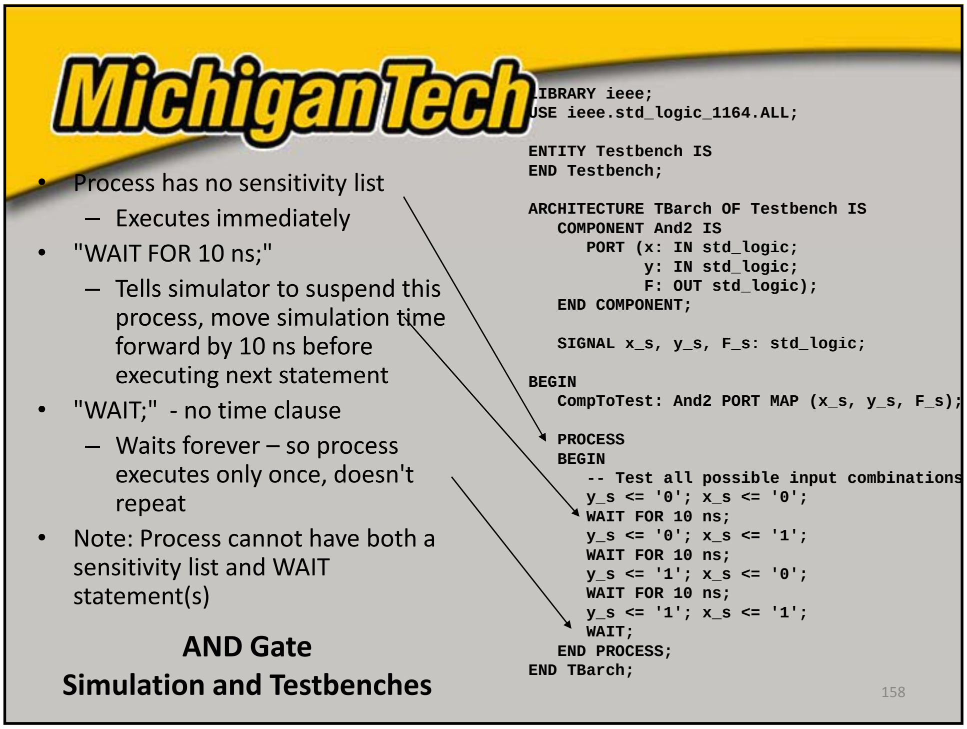


- HDL testbench
  - Entity with no ports
  - Declare component to test
  - Declare signal for each port
  - Instantiate component, map signals to ports
  - Set signal values at desired times

AND/OR/NOT Gates
Simulation and
Testbenches

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Testbench IS
END Testbench;

ARCHITECTURE TBarch OF Testbench IS
   COMPONENT And2 IS
      PORT (x: IN std_logic;
            y: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;

   SIGNAL x_s, y_s, F_s: std_logic;

BEGIN
   CompToTest: And2 PORT MAP (x_s, y_s, F_s);

   PROCESS
   BEGIN
      -- Test all possible input combinations
      y_s <= '0'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '0'; x_s <= '1';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '1';
      WAIT;
   END PROCESS;
END TBarch;
```

157

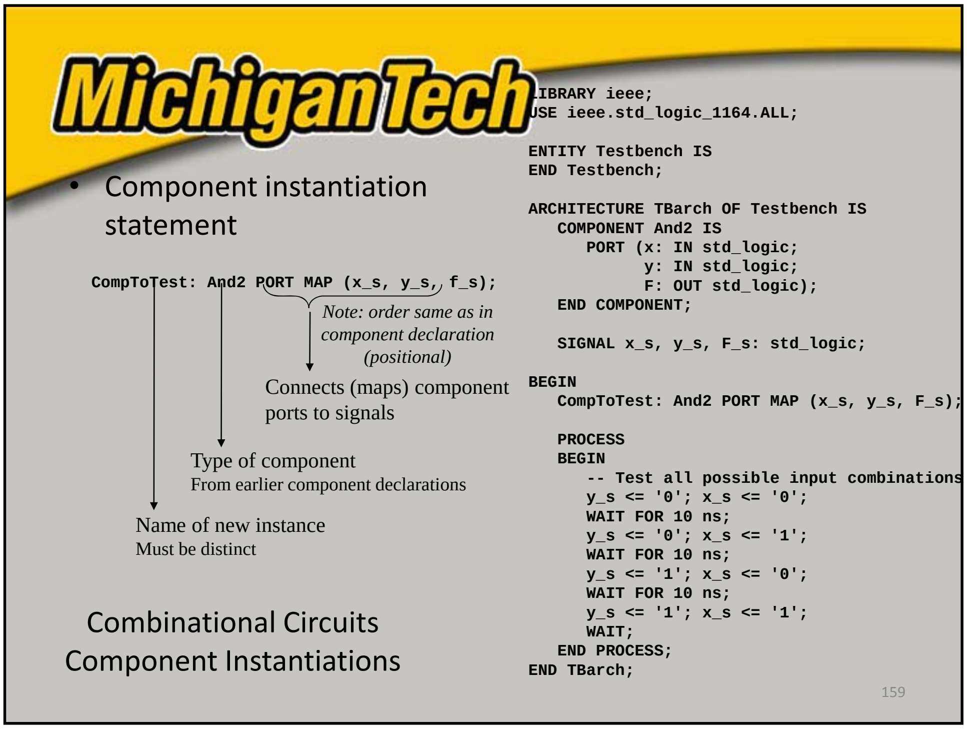


- Process has no sensitivity list
  - Executes immediately
- "WAIT FOR 10 ns;"
  - Tells simulator to suspend this process, move simulation time forward by 10 ns before executing next statement
- "WAIT;" - no time clause
  - Waits forever – so process executes only once, doesn't repeat
- Note: Process cannot have both a sensitivity list and WAIT statement(s)

AND Gate
Simulation and Testbenches

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Testbench IS
END Testbench;

ARCHITECTURE TBarch OF Testbench IS
   COMPONENT And2 IS
      PORT (x: IN std_logic;
            y: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;

   SIGNAL x_s, y_s, F_s: std_logic;

BEGIN
   CompToTest: And2 PORT MAP (x_s, y_s, F_s);

   PROCESS
   BEGIN
      -- Test all possible input combinations
      y_s <= '0'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '0'; x_s <= '1';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '1';
      WAIT;
   END PROCESS;
END TBarch;
```

158



- Component instantiation statement

```
CompToTest: And2 PORT MAP (x_s, y_s, f_s);
```

Note: order same as in component declaration (positional)

Connects (maps) component ports to signals

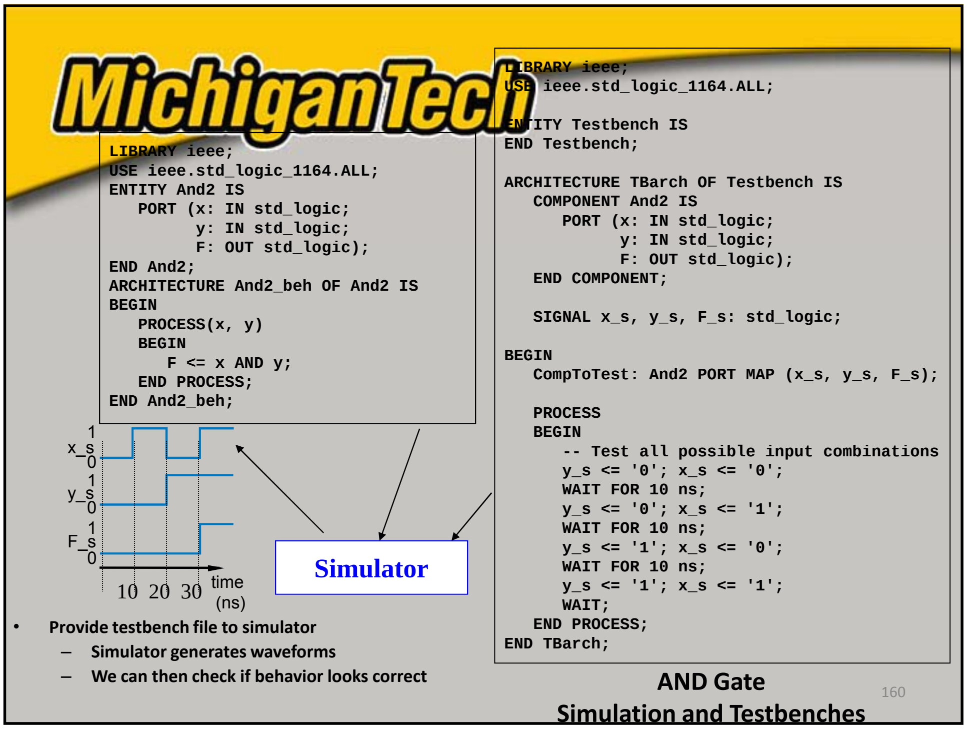
Type of component
From earlier component declarations

Name of new instance
Must be distinct

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Testbench IS
END Testbench;

ARCHITECTURE TBarch OF Testbench IS
   COMPONENT And2 IS
      PORT (x: IN std_logic;
            y: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;

   SIGNAL x_s, y_s, F_s: std_logic;

BEGIN
   CompToTest: And2 PORT MAP (x_s, y_s, F_s);

   PROCESS
   BEGIN
      -- Test all possible input combinations
      y_s <= '0'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '0'; x_s <= '1';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '1';
      WAIT;
   END PROCESS;
END TBarch;
```

Combinational Circuits
Component Instantiations

159



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY And2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END And2;
ARCHITECTURE And2_beh OF And2 IS
BEGIN
   PROCESS(x, y)
   BEGIN
      F <= x AND y;
   END PROCESS;
END And2_beh;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Testbench IS
END Testbench;

ARCHITECTURE TBarch OF Testbench IS
   COMPONENT And2 IS
      PORT (x: IN std_logic;
            y: IN std_logic;
            F: OUT std_logic);
   END COMPONENT;

   SIGNAL x_s, y_s, F_s: std_logic;

BEGIN
   CompToTest: And2 PORT MAP (x_s, y_s, F_s);

   PROCESS
   BEGIN
      -- Test all possible input combinations
      y_s <= '0'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '0'; x_s <= '1';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '0';
      WAIT FOR 10 ns;
      y_s <= '1'; x_s <= '1';
      WAIT;
   END PROCESS;
END TBarch;
```

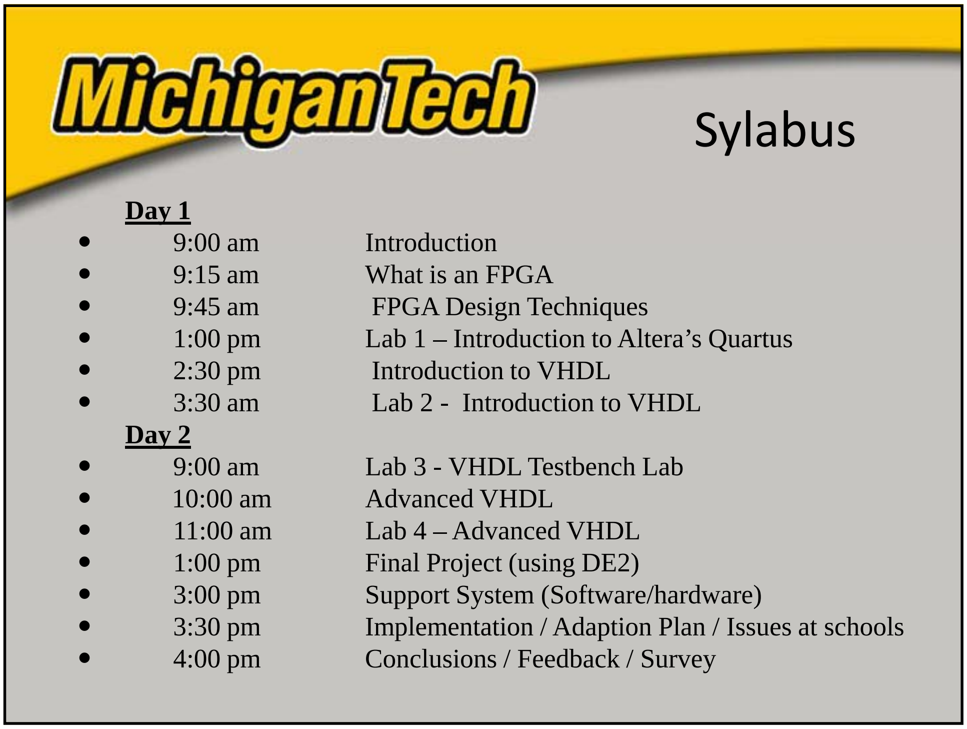
- Provide testbench file to simulator
  - Simulator generates waveforms
  - We can then check if behavior looks correct

AND Gate
Simulation and Testbenches

160

Please go to Lab2
Introduction to VHDL
in the Manual

161



Sylabus

Day 1

9:00 am	Introduction
9:15 am	What is an FPGA
9:45 am	FPGA Design Techniques
1:00 pm	Lab 1 – Introduction to Altera's Quartus
2:30 pm	Introduction to VHDL
3:30 am	Lab 2 - Introduction to VHDL

Day 2

9:00 am	Lab 3 - VHDL Testbench Lab
10:00 am	Advanced VHDL
11:00 am	Lab 4 – Advanced VHDL
1:00 pm	Final Project (using DE2)
3:00 pm	Support System (Software/hardware)
3:30 pm	Implementation / Adaption Plan / Issues at schools
4:00 pm	Conclusions / Feedback / Survey

MichiganTech

Please go to Lab3
VHDL Testbench
in the Manual

163

MichiganTech

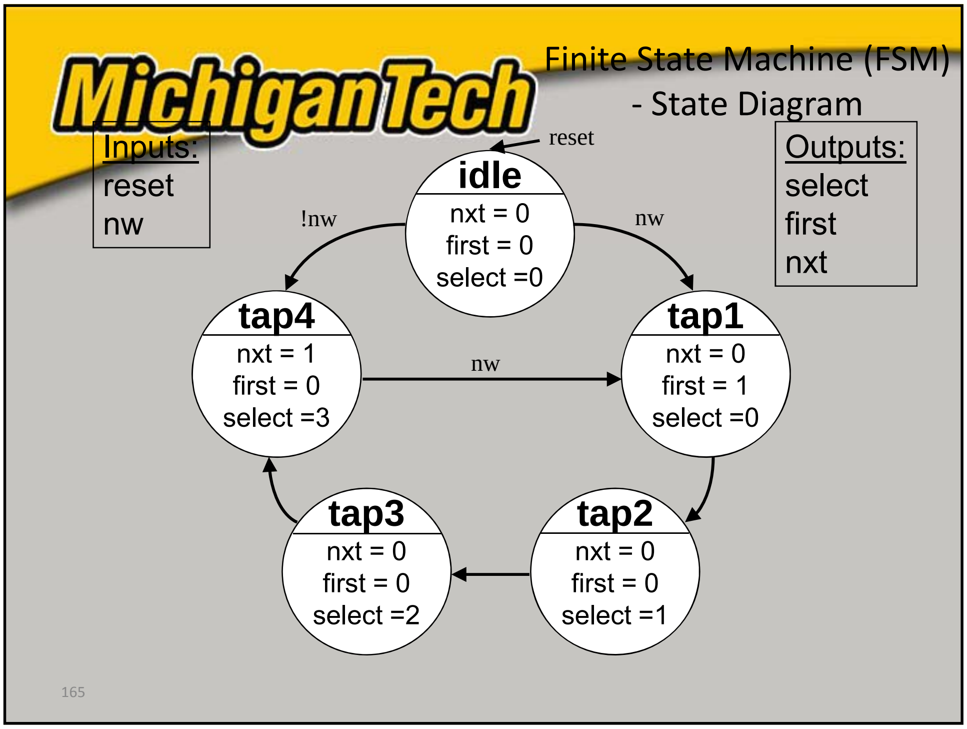
Advanced VHDL Design Techniques

FSM

Writing synthesizable VHDL

Create state machines and control their encoding

Optimize designs to improve resource usage and performance

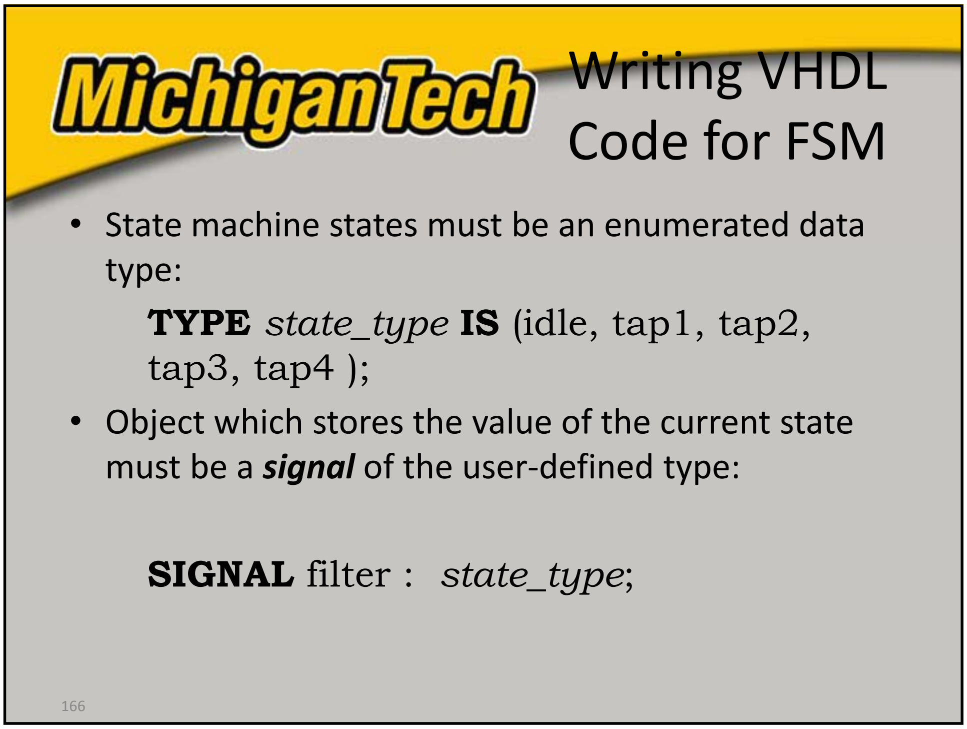


Finite State Machine (FSM) - State Diagram

Inputs:
reset
nw

Outputs:
select
first
nxt

165



Writing VHDL Code for FSM

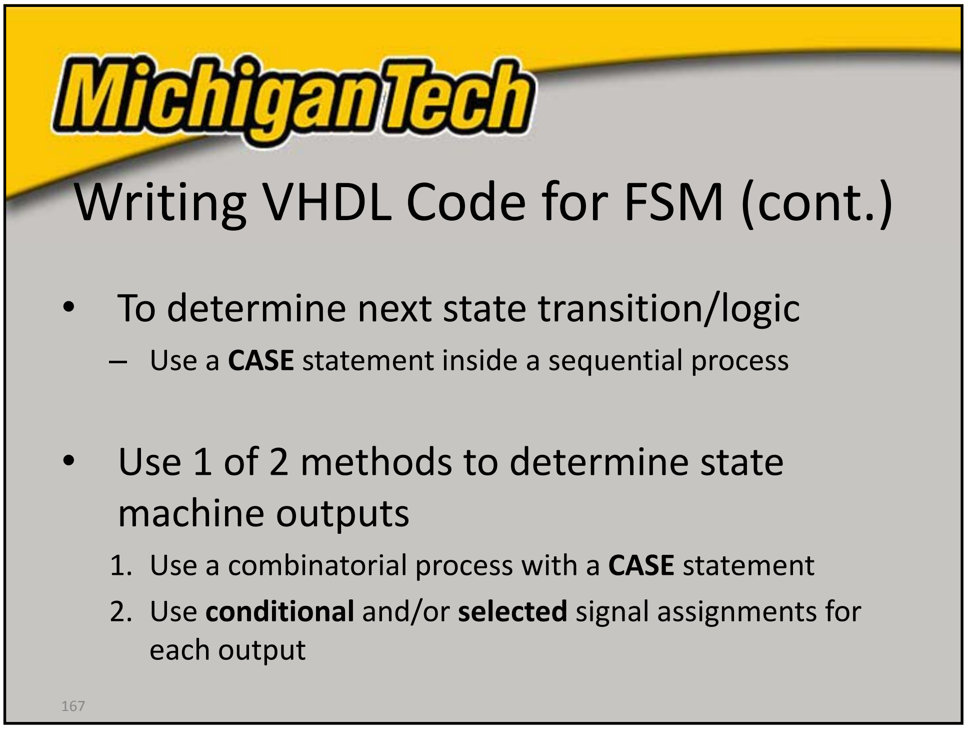
- State machine states must be an enumerated data type:

  TYPE *state_type* **IS** (idle, tap1, tap2, tap3, tap4);

- Object which stores the value of the current state must be a ***signal*** of the user-defined type:

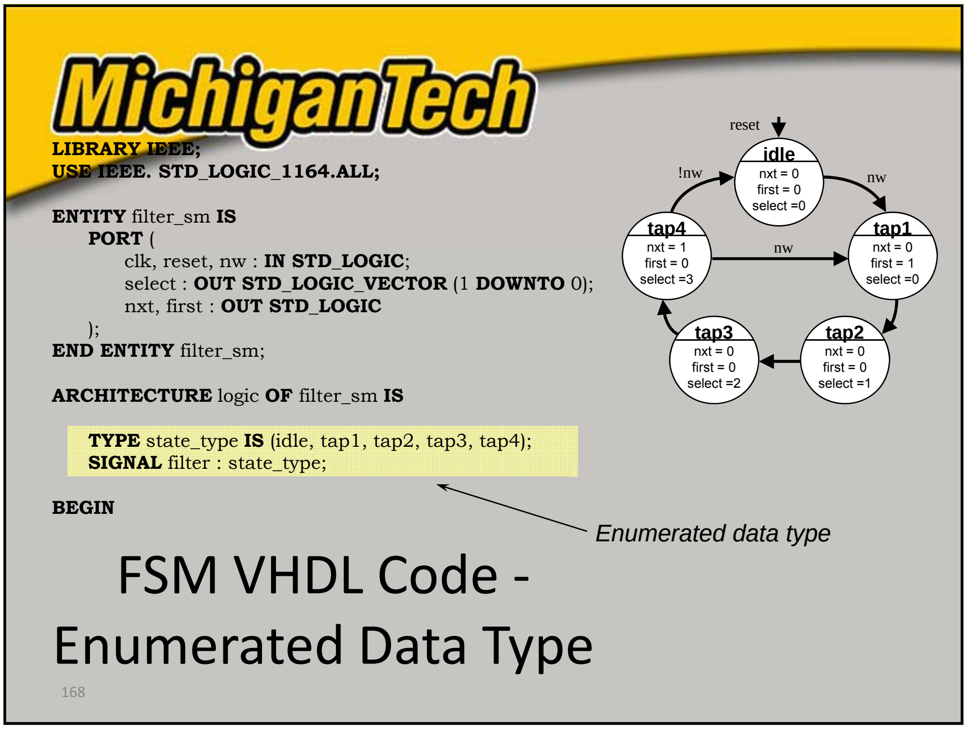
  SIGNAL filter : *state_type*;

166



Writing VHDL Code for FSM (cont.)

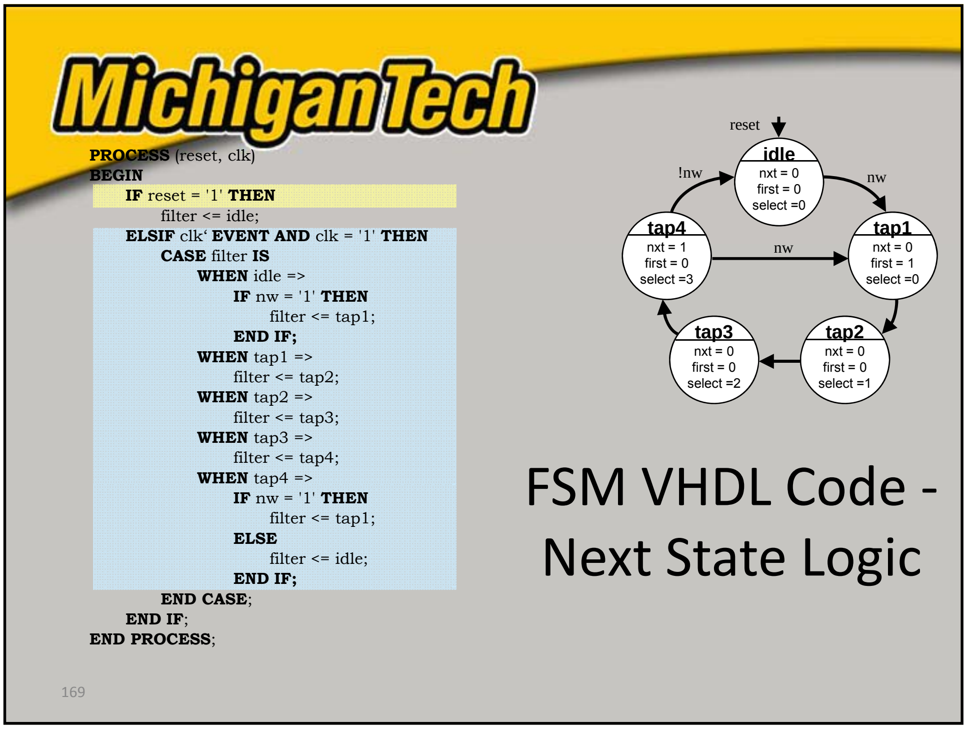
- To determine next state transition/logic
  - Use a **CASE** statement inside a sequential process
- Use 1 of 2 methods to determine state machine outputs
  1. Use a combinatorial process with a **CASE** statement
  2. Use **conditional** and/or **selected** signal assignments for each output



FSM VHDL Code - Enumerated Data Type

```
LIBRARY IEEE;
USE IEEE. STD_LOGIC_1164.ALL;

ENTITY filter_sm IS
     PORT (
          clk, reset, nw : IN STD_LOGIC;
          select : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
          nxt, first : OUT STD_LOGIC
     );
END ENTITY filter_sm;

ARCHITECTURE logic OF filter_sm IS

     TYPE state_type IS (idle, tap1, tap2, tap3, tap4);
     SIGNAL filter : state_type;

BEGIN
```

Enumerated data type

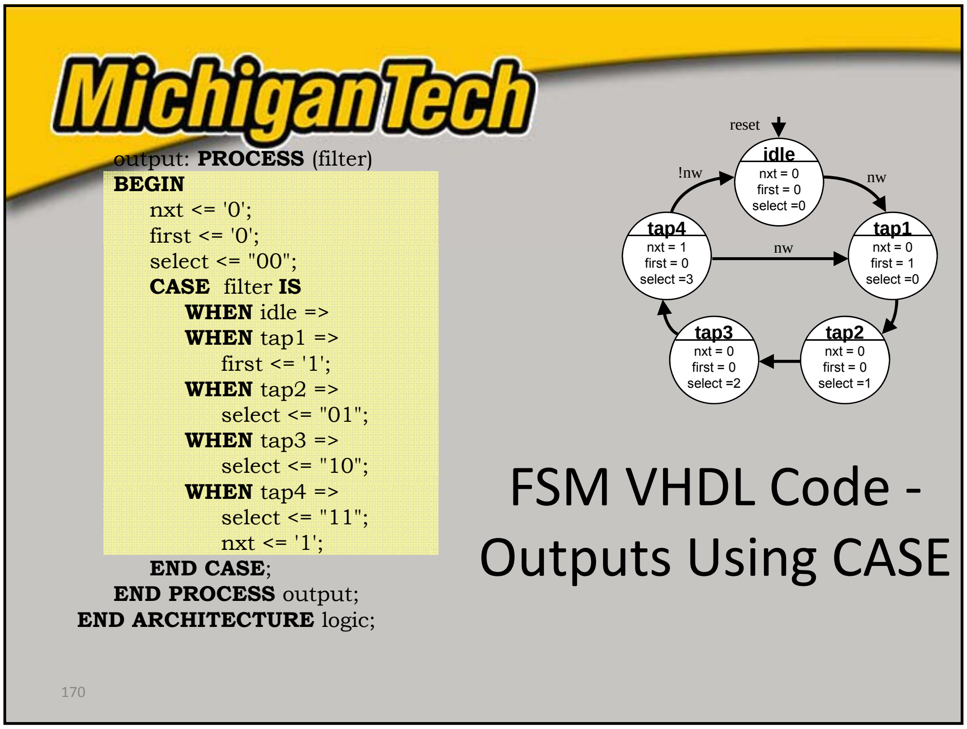


```
PROCESS (reset, clk)
BEGIN
    IF reset = '1' THEN
        filter <= idle;
    ELSIF clk' EVENT AND clk = '1' THEN
        CASE filter IS
            WHEN idle =>
                IF nw = '1' THEN
                    filter <= tap1;
                END IF;
            WHEN tap1 =>
                filter <= tap2;
            WHEN tap2 =>
                filter <= tap3;
            WHEN tap3 =>
                filter <= tap4;
            WHEN tap4 =>
                IF nw = '1' THEN
                    filter <= tap1;
                ELSE
                    filter <= idle;
                END IF;
        END CASE;
    END IF;
END PROCESS;
```

MichiganTech
reset
idle
nxt = 0
first = 0
select =0
!nw
nw
tap4
nxt = 1
first = 0
select =3
nw
tap1
nxt = 0
first = 1
select =0
tap3
nxt = 0
first = 0
select =2
tap2
nxt = 0
first = 0
select =1

FSM VHDL Code - Next State Logic

169

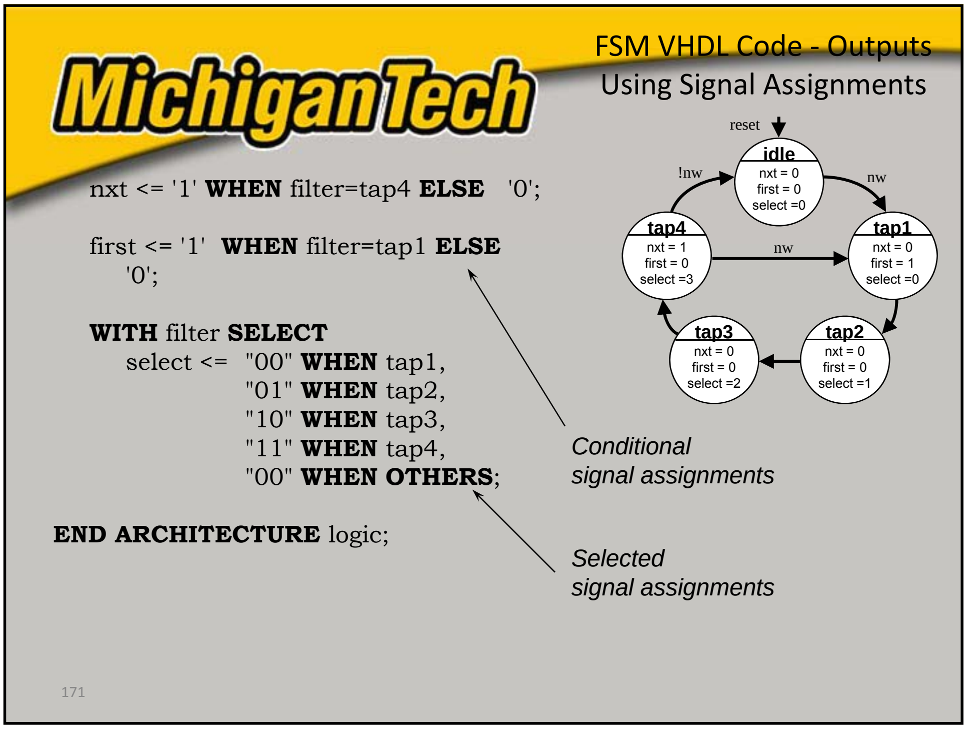


```
  output: PROCESS (filter)
  BEGIN
      nxt <= '0';
      first <= '0';
      select <= "00";
      CASE  filter IS
          WHEN idle =>
          WHEN tap1 =>
              first <= '1';
          WHEN tap2 =>
              select <= "01";
          WHEN tap3 =>
              select <= "10";
          WHEN tap4 =>
              select <= "11";
              nxt <= '1';
      END CASE;
  END PROCESS output;
END ARCHITECTURE logic;
```

MichiganTech
reset
idle
nxt = 0
first = 0
select =0
!nw
nw
tap4
nxt = 1
first = 0
select =3
nw
tap1
nxt = 0
first = 1
select =0
tap3
nxt = 0
first = 0
select =2
tap2
nxt = 0
first = 0
select =1

FSM VHDL Code - Outputs Using CASE

170



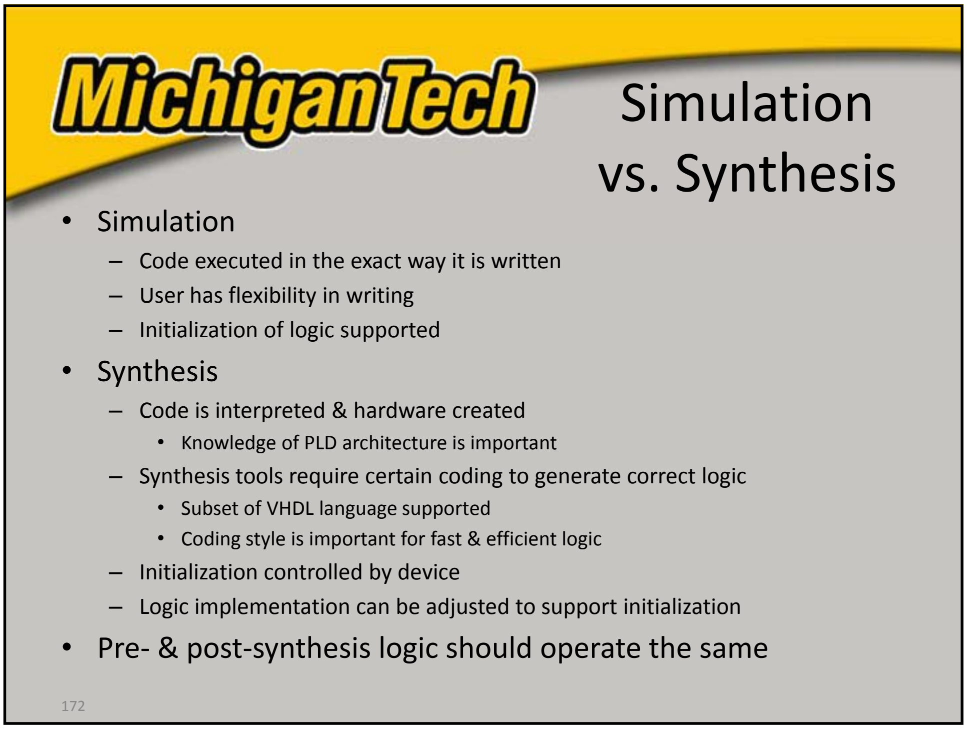
FSM VHDL Code - Outputs Using Signal Assignments

```
    nxt <= '1' WHEN filter=tap4 ELSE   '0';

    first <= '1'  WHEN filter=tap1 ELSE
        '0';

    WITH filter SELECT
        select <=  "00" WHEN tap1,
                   "01" WHEN tap2,
                   "10" WHEN tap3,
                   "11" WHEN tap4,
                   "00" WHEN OTHERS;

END ARCHITECTURE logic;
```

Conditional signal assignments

Selected signal assignments

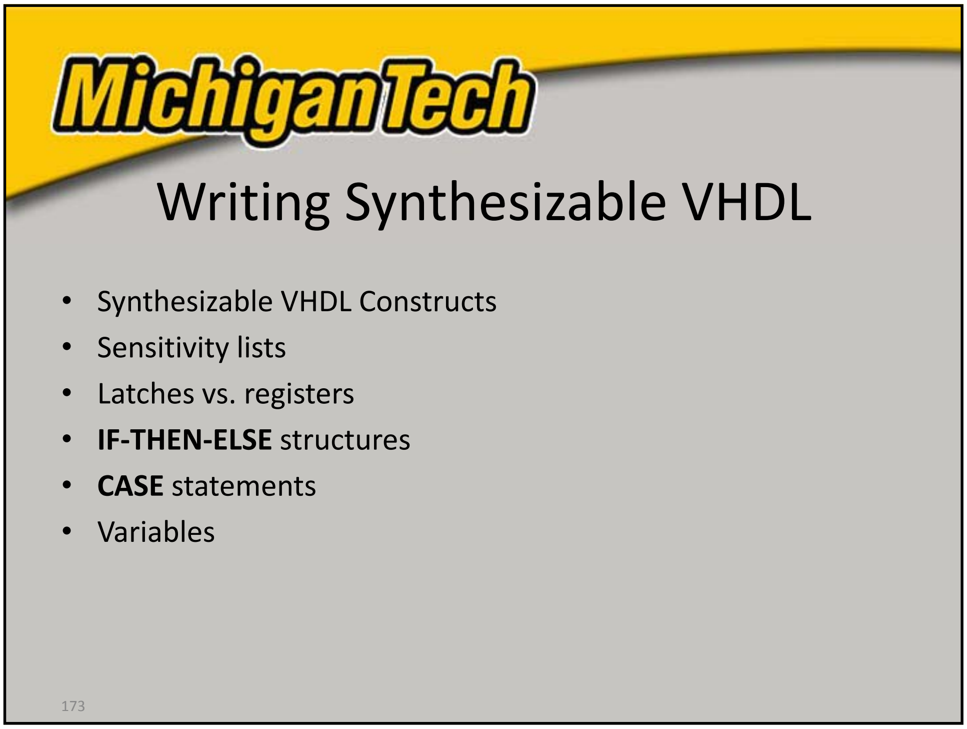
171



Simulation vs. Synthesis

- Simulation
  - Code executed in the exact way it is written
  - User has flexibility in writing
  - Initialization of logic supported
- Synthesis
  - Code is interpreted & hardware created
    - Knowledge of PLD architecture is important
  - Synthesis tools require certain coding to generate correct logic
    - Subset of VHDL language supported
    - Coding style is important for fast & efficient logic
  - Initialization controlled by device
  - Logic implementation can be adjusted to support initialization
- Pre- & post-synthesis logic should operate the same

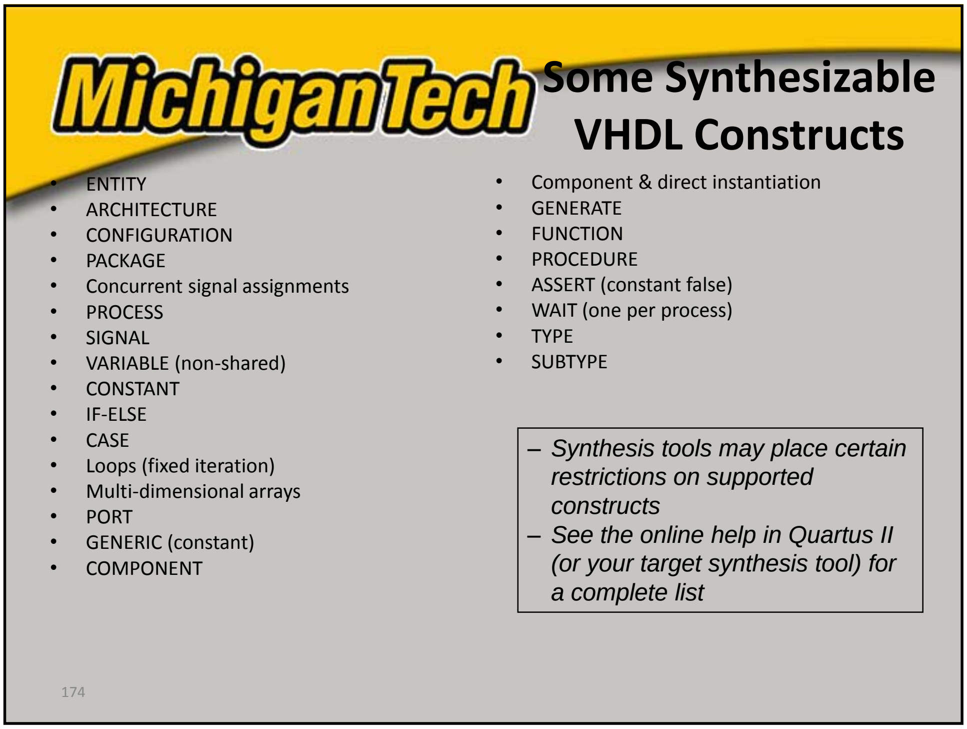
172



MichiganTech

Writing Synthesizable VHDL

- Synthesizable VHDL Constructs
- Sensitivity lists
- Latches vs. registers
- **IF-THEN-ELSE** structures
- **CASE** statements
- Variables



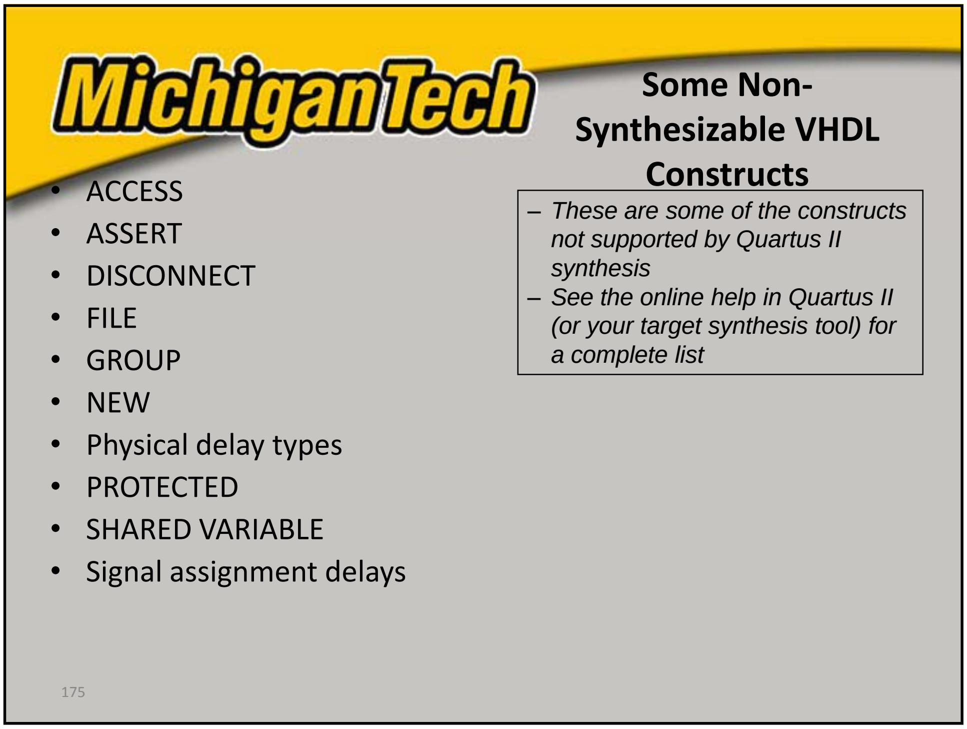
MichiganTech

Some Synthesizable VHDL Constructs

- ENTITY
- ARCHITECTURE
- CONFIGURATION
- PACKAGE
- Concurrent signal assignments
- PROCESS
- SIGNAL
- VARIABLE (non-shared)
- CONSTANT
- IF-ELSE
- CASE
- Loops (fixed iteration)
- Multi-dimensional arrays
- PORT
- GENERIC (constant)
- COMPONENT

- Component & direct instantiation
- GENERATE
- FUNCTION
- PROCEDURE
- ASSERT (constant false)
- WAIT (one per process)
- TYPE
- SUBTYPE

– *Synthesis tools may place certain restrictions on supported constructs*
– *See the online help in Quartus II (or your target synthesis tool) for a complete list*

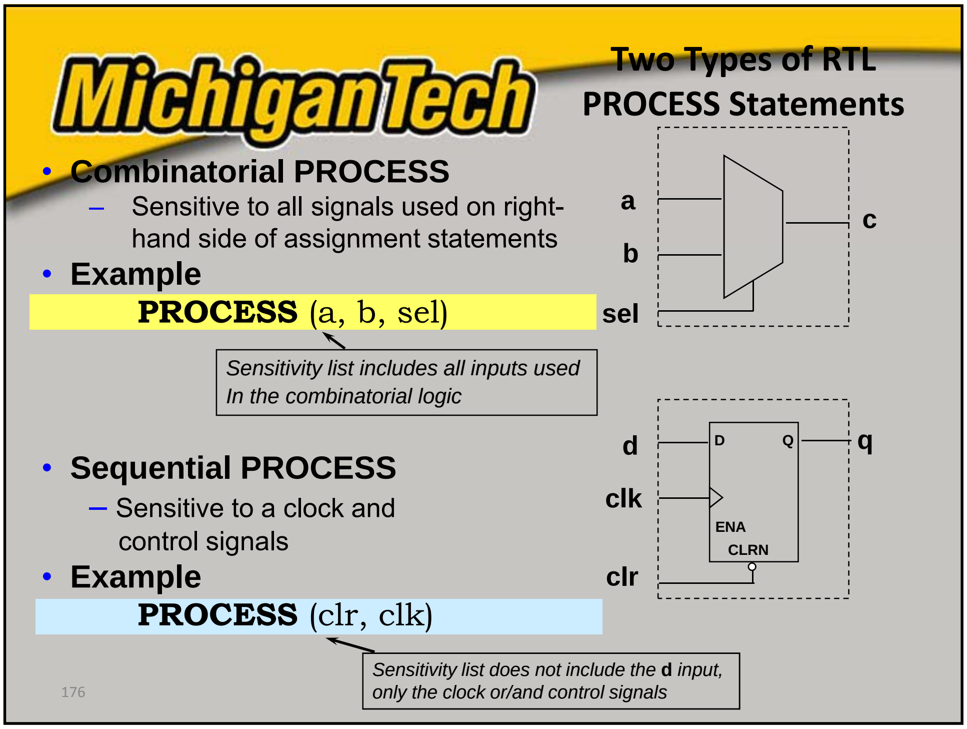


Some Non-Synthesizable VHDL Constructs

- ACCESS
- ASSERT
- DISCONNECT
- FILE
- GROUP
- NEW
- Physical delay types
- PROTECTED
- SHARED VARIABLE
- Signal assignment delays

- *These are some of the constructs not supported by Quartus II synthesis*
- *See the online help in Quartus II (or your target synthesis tool) for a complete list*

175



Two Types of RTL PROCESS Statements

- **Combinatorial PROCESS**
  - Sensitive to all signals used on right-hand side of assignment statements
- **Example**

PROCESS (a, b, sel)

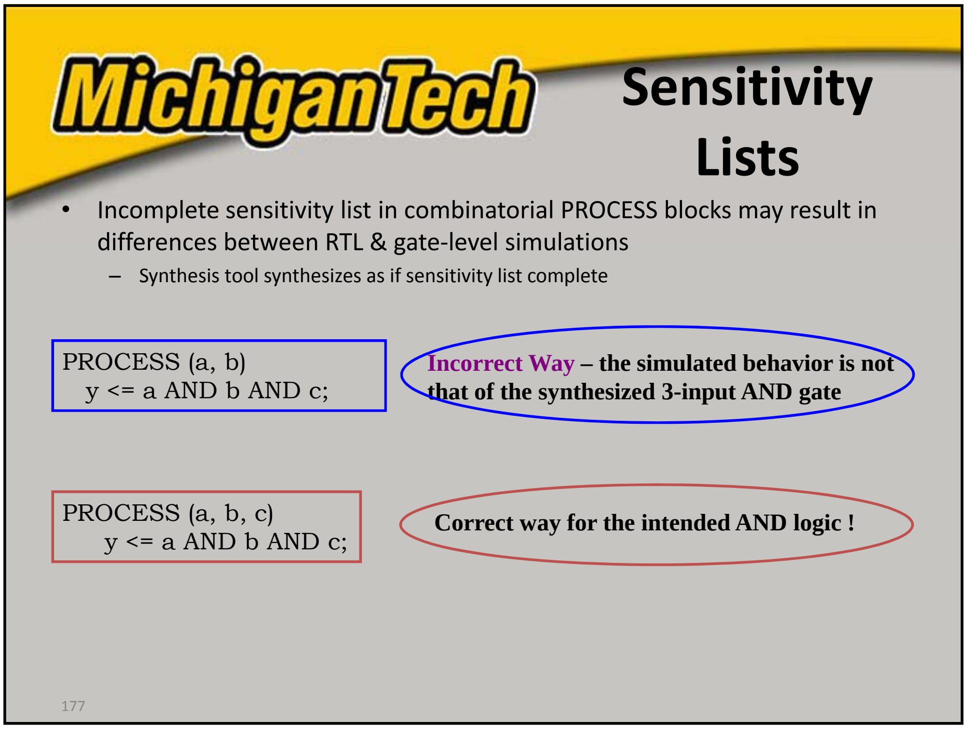
Sensitivity list includes all inputs used In the combinatorial logic

- **Sequential PROCESS**
  - Sensitive to a clock and control signals
- **Example**

PROCESS (clr, clk)

Sensitivity list does not include the **d** *input, only the clock or/and control signals*

176



MichiganTech

Sensitivity Lists

- Incomplete sensitivity list in combinatorial PROCESS blocks may result in differences between RTL & gate-level simulations
  - Synthesis tool synthesizes as if sensitivity list complete

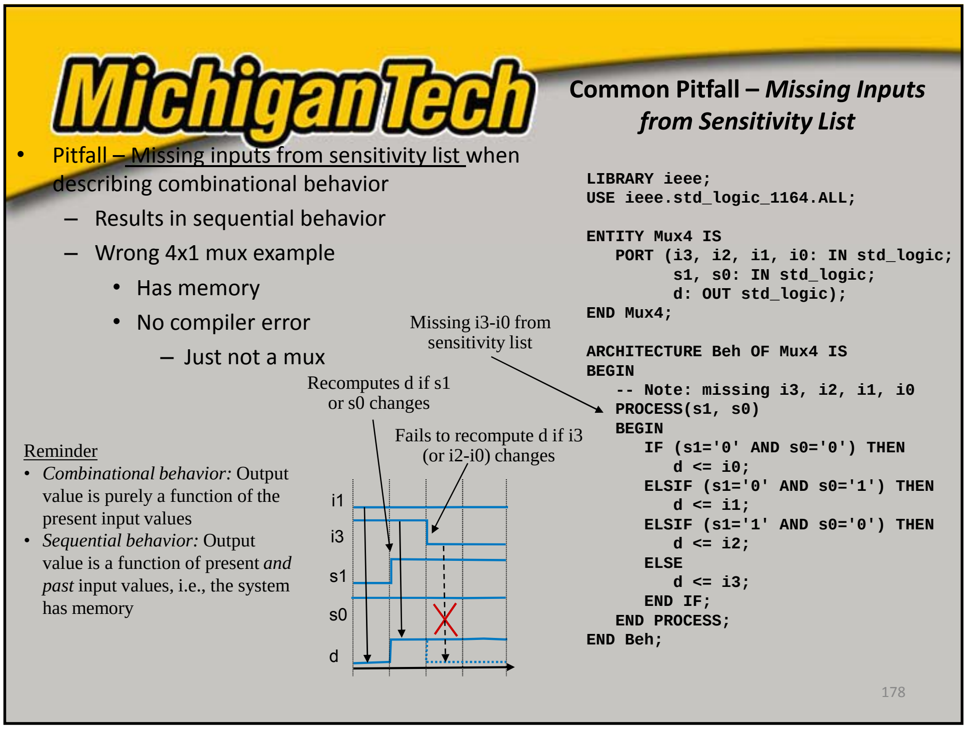
```
PROCESS (a, b)
  y <= a AND b AND c;
```

Incorrect Way – the simulated behavior is not that of the synthesized 3-input AND gate

```
PROCESS (a, b, c)
    y <= a AND b AND c;
```

Correct way for the intended AND logic !

177



MichiganTech

Common Pitfall – *Missing Inputs from Sensitivity List*

- Pitfall – Missing inputs from sensitivity list when describing combinational behavior
  - Results in sequential behavior
  - Wrong 4x1 mux example
    - Has memory
    - No compiler error
      - Just not a mux

Reminder

- *Combinational behavior:* Output value is purely a function of the present input values
- *Sequential behavior:* Output value is a function of present *and past* input values, i.e., the system has memory

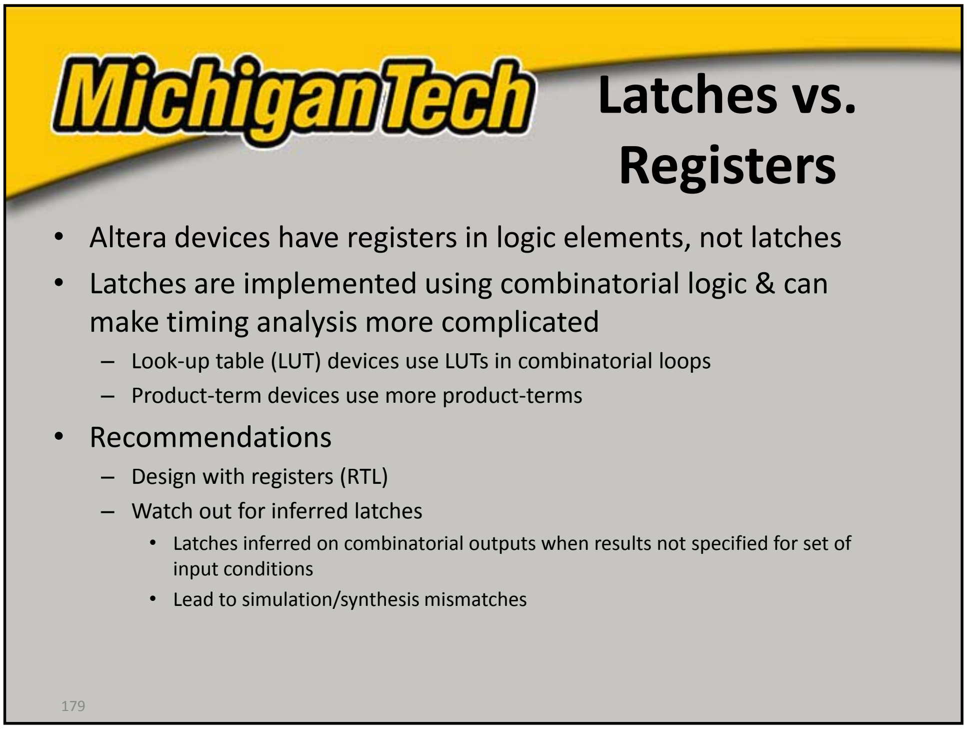
Missing i3-i0 from sensitivity list

Recomputes d if s1 or s0 changes

Fails to recompute d if i3 (or i2-i0) changes

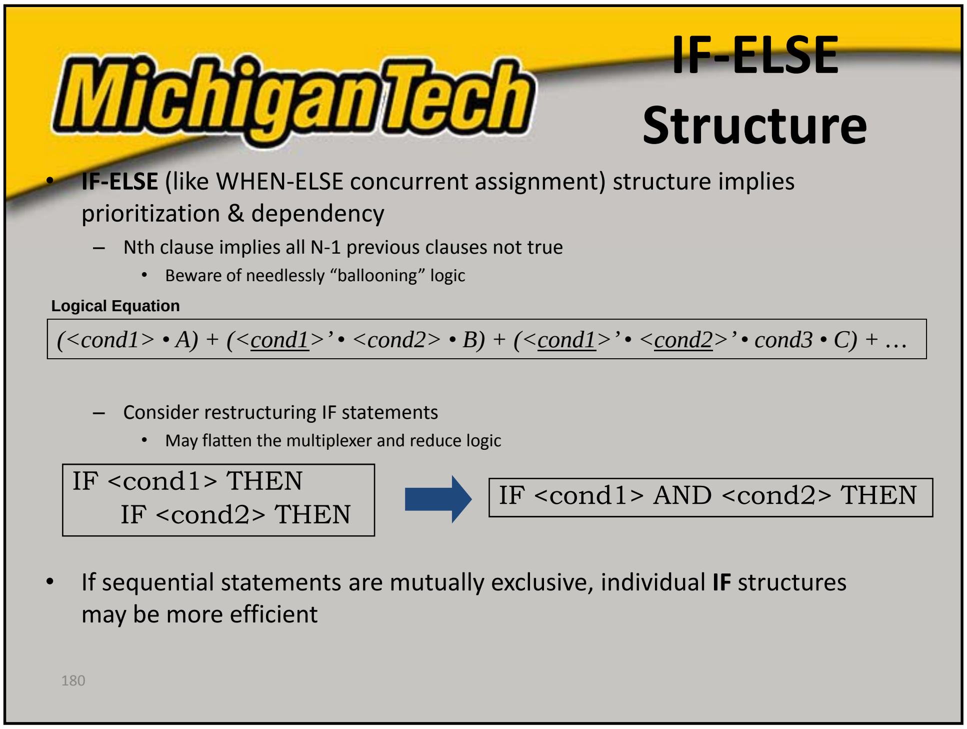
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Mux4 IS
   PORT (i3, i2, i1, i0: IN std_logic;
         s1, s0: IN std_logic;
         d: OUT std_logic);
END Mux4;

ARCHITECTURE Beh OF Mux4 IS
BEGIN
   -- Note: missing i3, i2, i1, i0
   PROCESS(s1, s0)
   BEGIN
      IF (s1='0' AND s0='0') THEN
         d <= i0;
      ELSIF (s1='0' AND s0='1') THEN
         d <= i1;
      ELSIF (s1='1' AND s0='0') THEN
         d <= i2;
      ELSE
         d <= i3;
      END IF;
   END PROCESS;
END Beh;
```

178



Latches vs. Registers

- Altera devices have registers in logic elements, not latches
- Latches are implemented using combinatorial logic & can make timing analysis more complicated
  - Look-up table (LUT) devices use LUTs in combinatorial loops
  - Product-term devices use more product-terms
- Recommendations
  - Design with registers (RTL)
  - Watch out for inferred latches
    - Latches inferred on combinatorial outputs when results not specified for set of input conditions
    - Lead to simulation/synthesis mismatches



IF-ELSE Structure

- **IF-ELSE** (like WHEN-ELSE concurrent assignment) structure implies prioritization & dependency
  - Nth clause implies all N-1 previous clauses not true
    - Beware of needlessly "ballooning" logic

Logical Equation

$$(\langle cond1\rangle \cdot A) + (\langle \underline{cond1}\rangle' \cdot \langle cond2\rangle \cdot B) + (\langle \underline{cond1}\rangle' \cdot \langle \underline{cond2}\rangle' \cdot cond3 \cdot C) + \ldots$$

  - Consider restructuring IF statements
    - May flatten the multiplexer and reduce logic

```
IF <cond1> THEN
    IF <cond2> THEN
```

→

```
IF <cond1> AND <cond2> THEN
```

- If sequential statements are mutually exclusive, individual **IF** structures may be more efficient

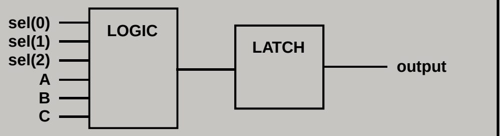
When Writing IF-ELSE Structures…

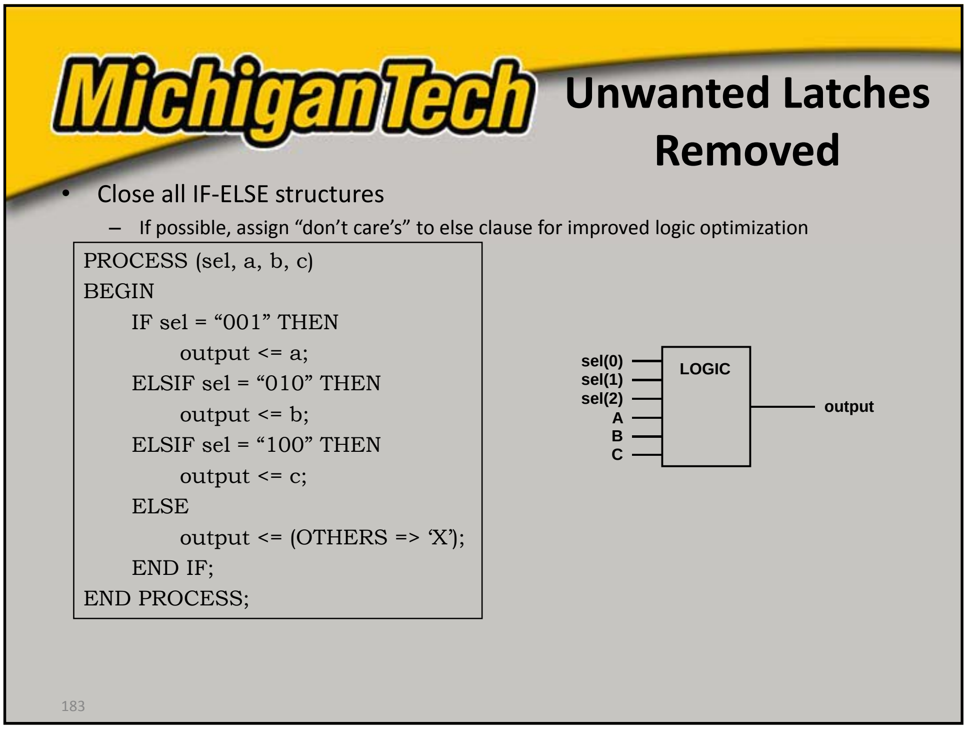
- Cover all cases
  - Uncovered cases in combinatorial processes result in latches
- For efficiency, consider
  - Using don’t cares (‘-’ or ‘X’) for final ELSE clause (avoiding unnecessary default conditions)
    - Synthesis tool has freedom to encode don’t cares for maximum optimization
  - Assigning initial values and explicitly covering only those results different from initial values

Unwanted Latches

- Combinatorial processes that do not cover all possible input conditions generate latches

```
PROCESS (sel, a, b, c)
BEGIN
    IF sel = “001” THEN
        output <= a;
    ELSIF sel = “010” THEN
        output <= b;
    ELSIF sel = “100” THEN
        output <= c;
    END IF;
END PROCESS;
```



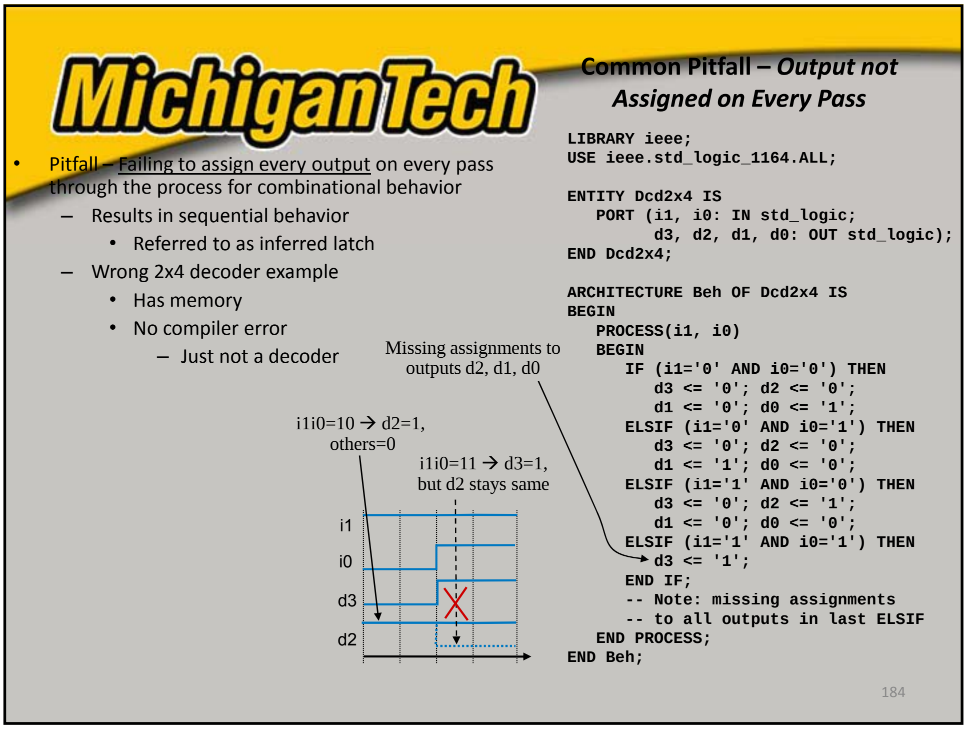


Unwanted Latches Removed

- Close all IF-ELSE structures
  - If possible, assign “don’t care’s” to else clause for improved logic optimization

```
PROCESS (sel, a, b, c)
BEGIN
    IF sel = “001” THEN
        output <= a;
    ELSIF sel = “010” THEN
        output <= b;
    ELSIF sel = “100” THEN
        output <= c;
    ELSE
        output <= (OTHERS => ‘X’);
    END IF;
END PROCESS;
```

sel(0)
sel(1)
sel(2)
A
B
C
LOGIC
output



Common Pitfall – *Output not Assigned on Every Pass*

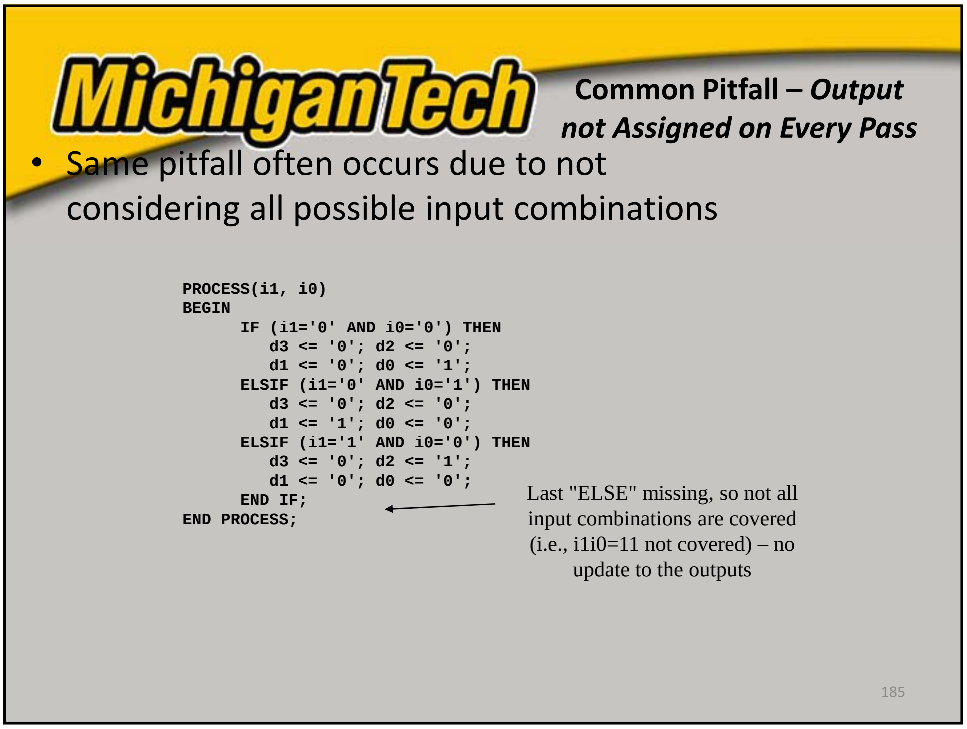
- Pitfall – Failing to assign every output on every pass through the process for combinational behavior
  - Results in sequential behavior
    - Referred to as inferred latch
  - Wrong 2x4 decoder example
    - Has memory
    - No compiler error
      - Just not a decoder

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Dcd2x4 IS
   PORT (i1, i0: IN std_logic;
         d3, d2, d1, d0: OUT std_logic);
END Dcd2x4;

ARCHITECTURE Beh OF Dcd2x4 IS
BEGIN
   PROCESS(i1, i0)
   BEGIN
      IF (i1='0' AND i0='0') THEN
         d3 <= '0'; d2 <= '0';
         d1 <= '0'; d0 <= '1';
      ELSIF (i1='0' AND i0='1') THEN
         d3 <= '0'; d2 <= '0';
         d1 <= '1'; d0 <= '0';
      ELSIF (i1='1' AND i0='0') THEN
         d3 <= '0'; d2 <= '1';
         d1 <= '0'; d0 <= '0';
      ELSIF (i1='1' AND i0='1') THEN
         d3 <= '1';
      END IF;
      -- Note: missing assignments
      -- to all outputs in last ELSIF
   END PROCESS;
END Beh;
```

Missing assignments to outputs d2, d1, d0

i1i0=10 → d2=1, others=0

i1i0=11 → d3=1, but d2 stays same



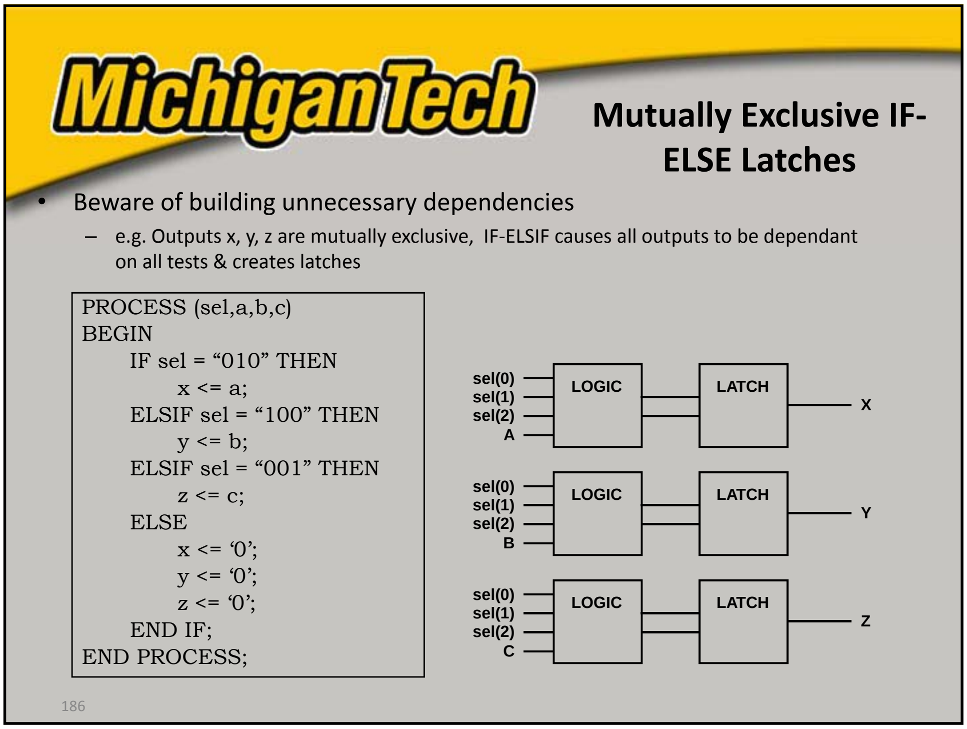
Common Pitfall – *Output not Assigned on Every Pass*

- Same pitfall often occurs due to not considering all possible input combinations

```
PROCESS(i1, i0)
BEGIN
      IF (i1='0' AND i0='0') THEN
         d3 <= '0'; d2 <= '0';
         d1 <= '0'; d0 <= '1';
      ELSIF (i1='0' AND i0='1') THEN
         d3 <= '0'; d2 <= '0';
         d1 <= '1'; d0 <= '0';
      ELSIF (i1='1' AND i0='0') THEN
         d3 <= '0'; d2 <= '1';
         d1 <= '0'; d0 <= '0';
      END IF;
END PROCESS;
```

Last "ELSE" missing, so not all input combinations are covered (i.e., i1i0=11 not covered) – no update to the outputs

185

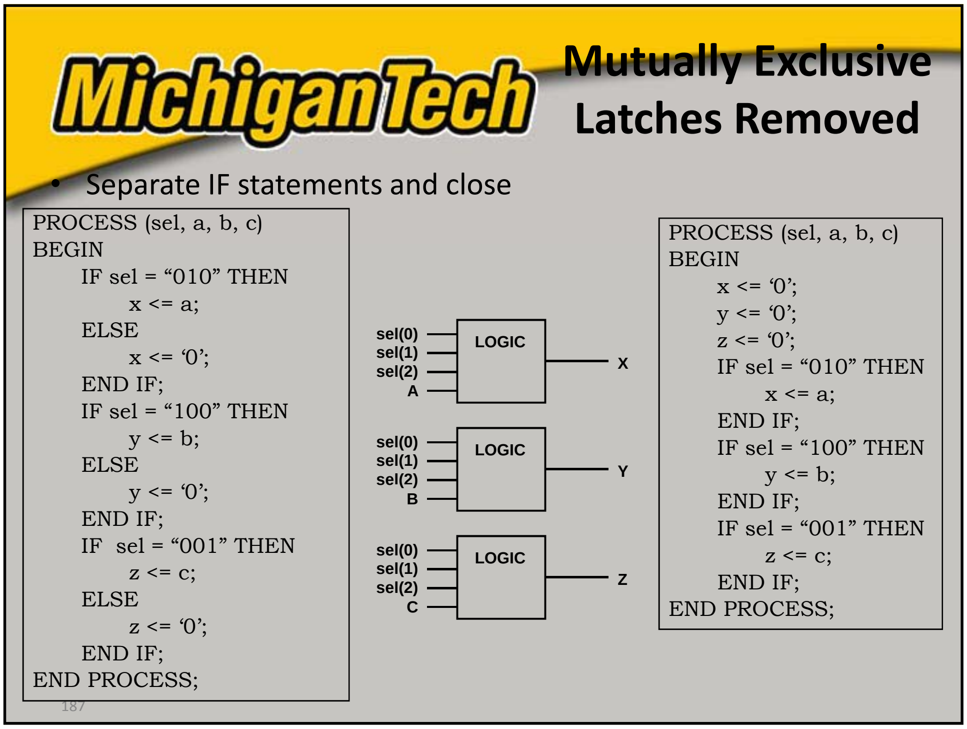


Mutually Exclusive IF-ELSE Latches

- Beware of building unnecessary dependencies
  - e.g. Outputs x, y, z are mutually exclusive, IF-ELSIF causes all outputs to be dependant on all tests & creates latches

```
PROCESS (sel,a,b,c)
BEGIN
      IF sel = “010” THEN
            x <= a;
      ELSIF sel = “100” THEN
            y <= b;
      ELSIF sel = “001” THEN
            z <= c;
      ELSE
            x <= ‘0’;
            y <= ‘0’;
            z <= ‘0’;
      END IF;
END PROCESS;
```

186

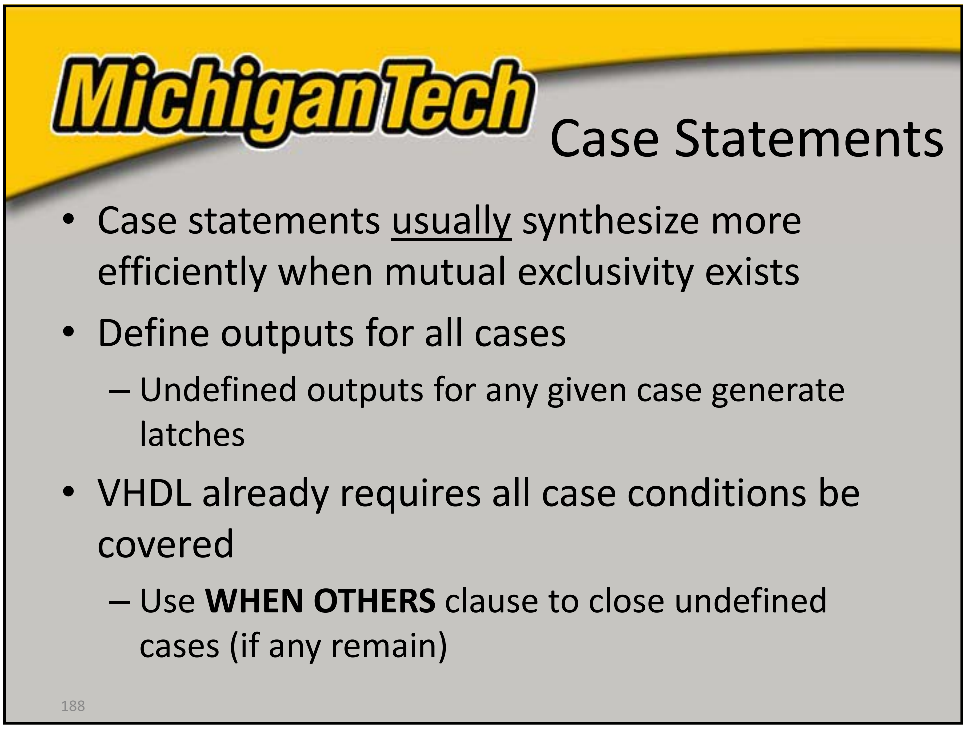


Mutually Exclusive Latches Removed

- Separate IF statements and close

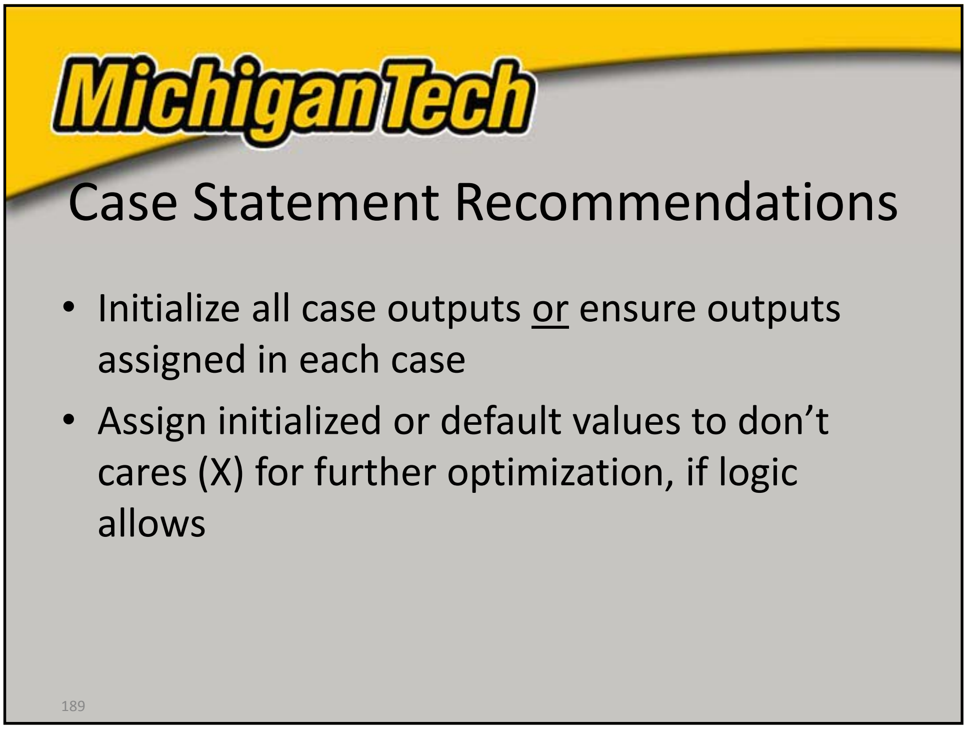
```
PROCESS (sel, a, b, c)
BEGIN
        IF sel = "010" THEN
                x <= a;
        ELSE
                x <= '0';
        END IF;
        IF sel = "100" THEN
                y <= b;
        ELSE
                y <= '0';
        END IF;
        IF  sel = "001" THEN
                z <= c;
        ELSE
                z <= '0';
        END IF;
END PROCESS;
```

```
PROCESS (sel, a, b, c)
BEGIN
        x <= '0';
        y <= '0';
        z <= '0';
        IF sel = "010" THEN
                x <= a;
        END IF;
        IF sel = "100" THEN
                y <= b;
        END IF;
        IF sel = "001" THEN
                z <= c;
        END IF;
END PROCESS;
```



Case Statements

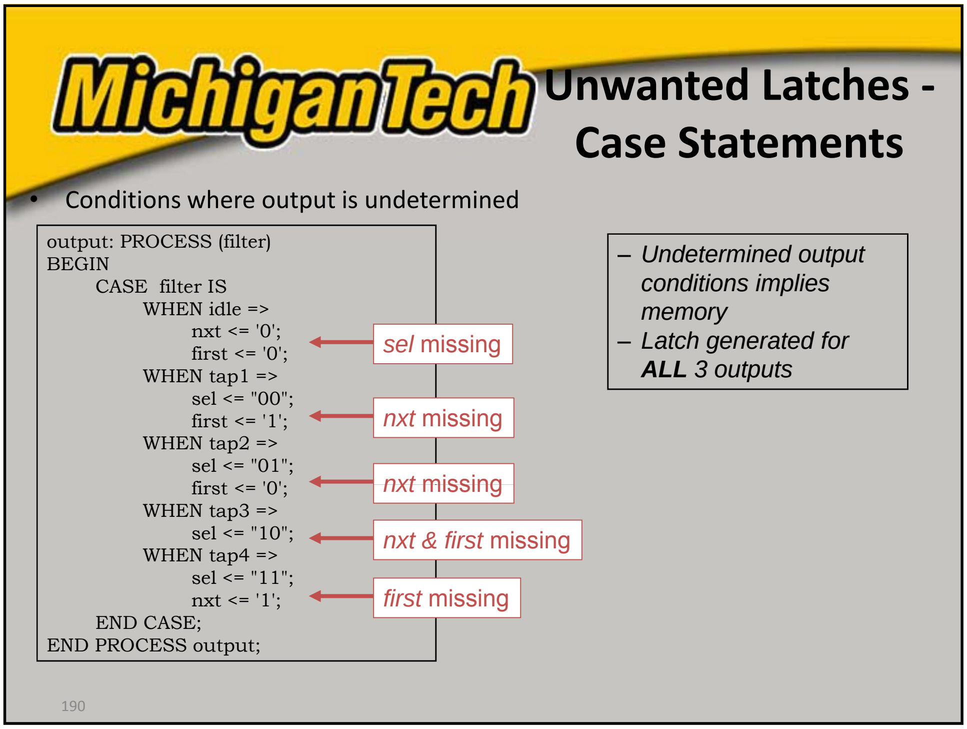
- Case statements usually synthesize more efficiently when mutual exclusivity exists
- Define outputs for all cases
  - Undefined outputs for any given case generate latches
- VHDL already requires all case conditions be covered
  - Use **WHEN OTHERS** clause to close undefined cases (if any remain)



MichiganTech

Case Statement Recommendations

- Initialize all case outputs or ensure outputs assigned in each case
- Assign initialized or default values to don’t cares (X) for further optimization, if logic allows



MichiganTech

Unwanted Latches - Case Statements

- Conditions where output is undetermined

```
output: PROCESS (filter)
BEGIN
      CASE  filter IS
            WHEN idle =>
                  nxt <= '0';
                  first <= '0';
            WHEN tap1 =>
                  sel <= "00";
                  first <= '1';
            WHEN tap2 =>
                  sel <= "01";
                  first <= '0';
            WHEN tap3 =>
                  sel <= "10";
            WHEN tap4 =>
                  sel <= "11";
                  nxt <= '1';
      END CASE;
END PROCESS output;
```

sel missing

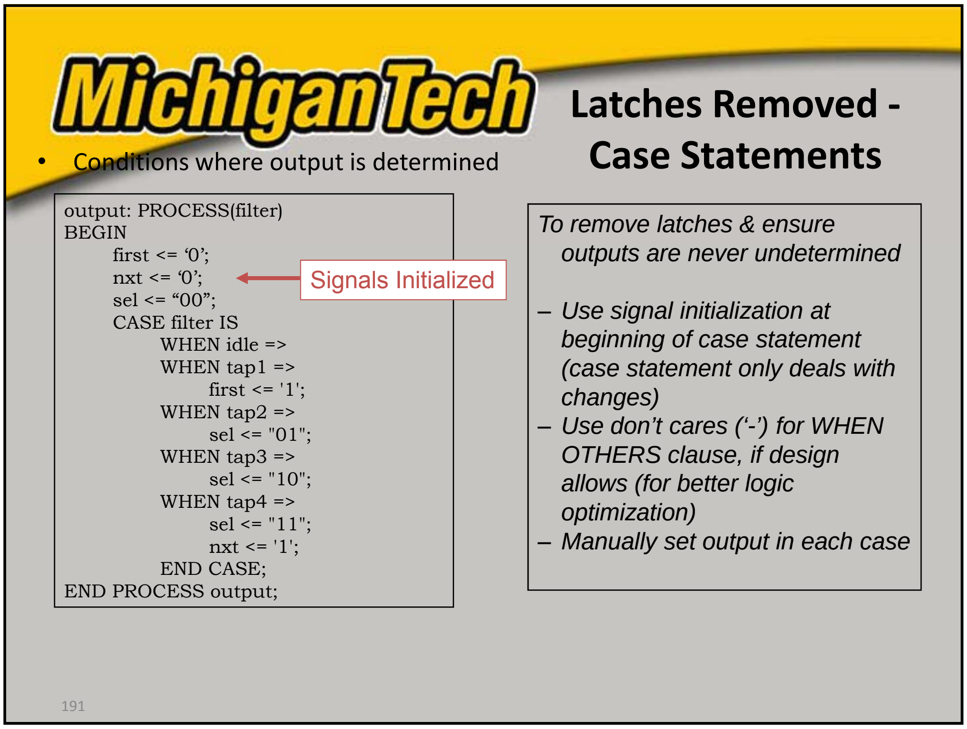
nxt missing

nxt missing

nxt & first missing

first missing

- *Undetermined output conditions implies memory*
- *Latch generated for **ALL** 3 outputs*



Latches Removed - Case Statements

- Conditions where output is determined

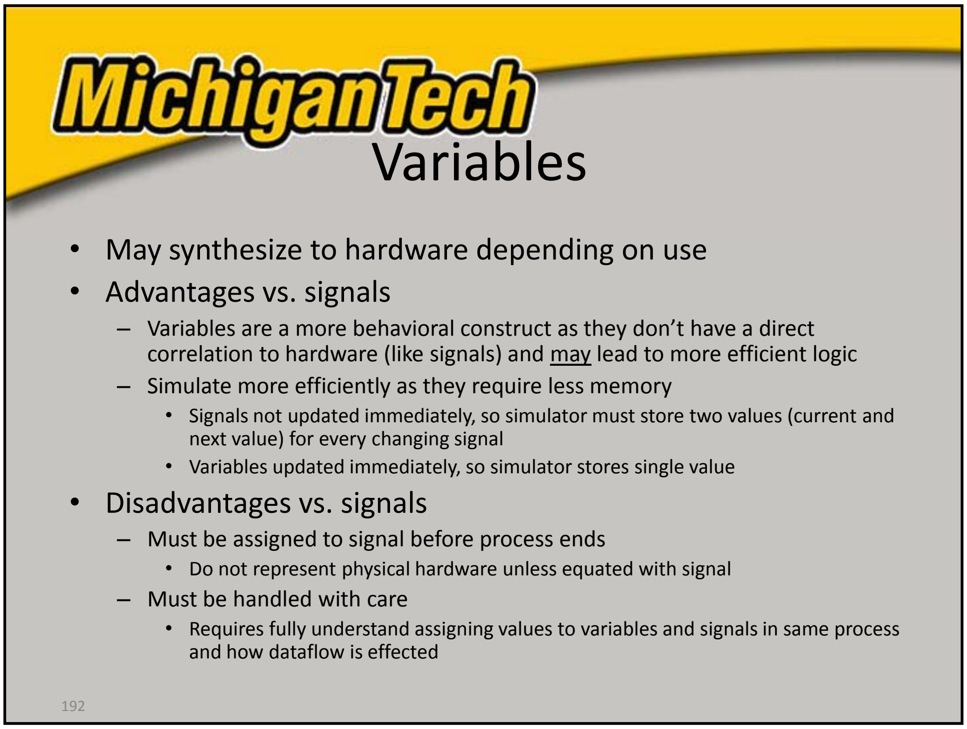
```
output: PROCESS(filter)
BEGIN
        first <= ‘0’;
        nxt <= ‘0’;
        sel <= “00”;
        CASE filter IS
                WHEN idle =>
                WHEN tap1 =>
                        first <= '1';
                WHEN tap2 =>
                        sel <= "01";
                WHEN tap3 =>
                        sel <= "10";
                WHEN tap4 =>
                        sel <= "11";
                        nxt <= '1';
                END CASE;
END PROCESS output;
```

Signals Initialized

To remove latches & ensure outputs are never undetermined

- *Use signal initialization at beginning of case statement (case statement only deals with changes)*
- *Use don’t cares (‘-’) for WHEN OTHERS clause, if design allows (for better logic optimization)*
- *Manually set output in each case*

191



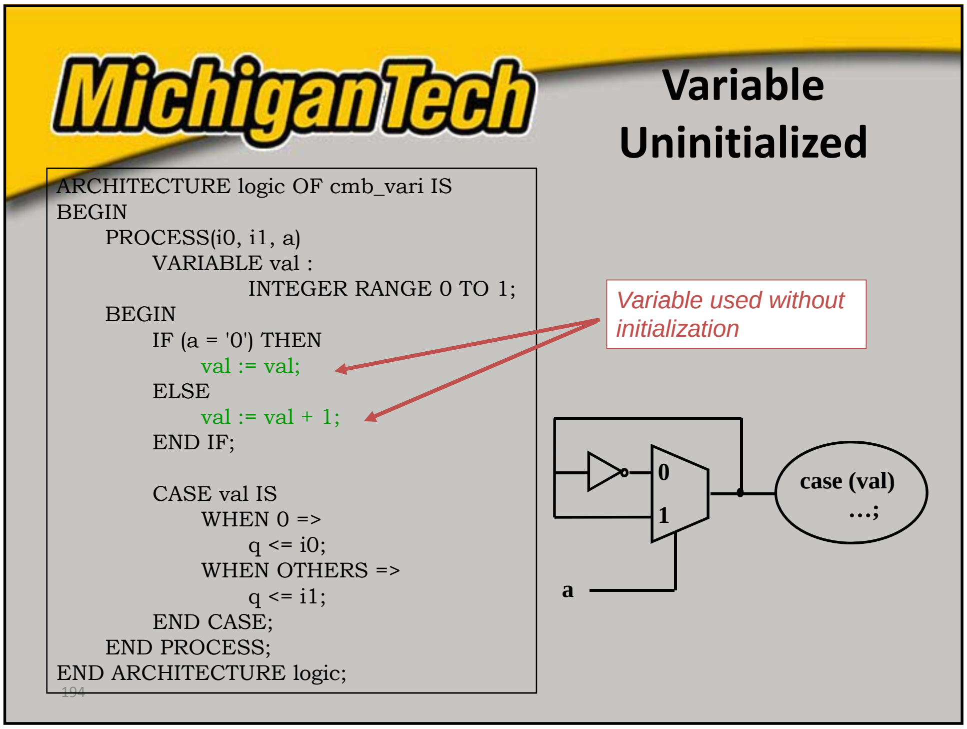
Variables

- May synthesize to hardware depending on use
- Advantages vs. signals
  - Variables are a more behavioral construct as they don’t have a direct correlation to hardware (like signals) and may lead to more efficient logic
  - Simulate more efficiently as they require less memory
    - Signals not updated immediately, so simulator must store two values (current and next value) for every changing signal
    - Variables updated immediately, so simulator stores single value
- Disadvantages vs. signals
  - Must be assigned to signal before process ends
    - Do not represent physical hardware unless equated with signal
  - Must be handled with care
    - Requires fully understand assigning values to variables and signals in same process and how dataflow is effected

192

Variables & Latches (Recommendations)

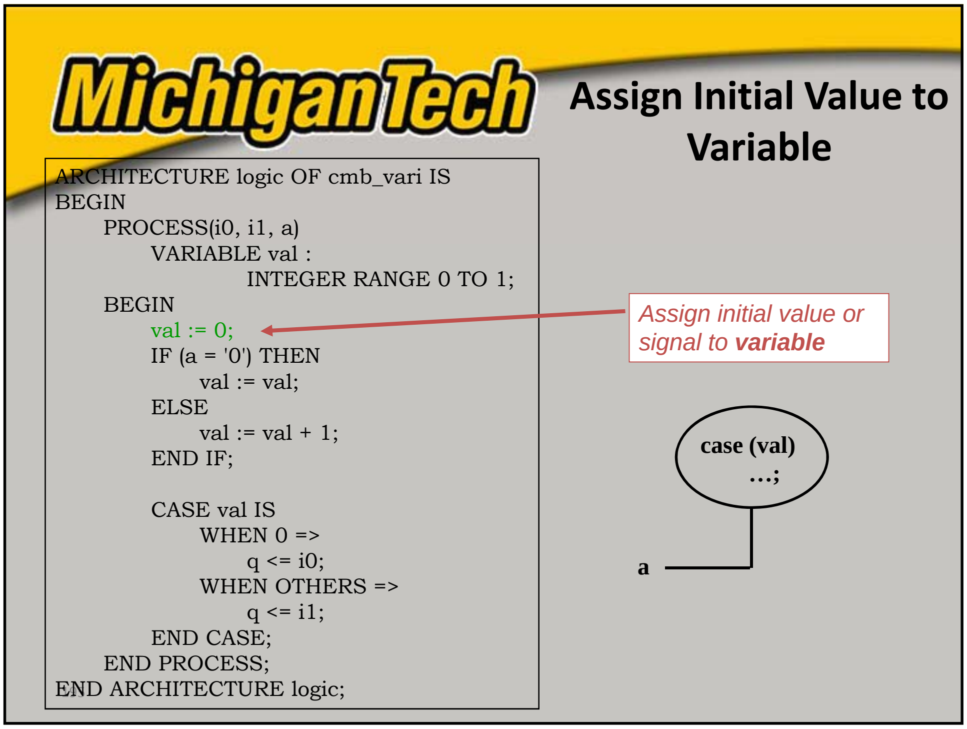
- Assign an initial value or signal to a variable unless feedback is desired
- If a variable is not assigned an initial value or signal in a combinatorial process, a latch will be generated
  - This could cause your design to not function as intended



Variable Uninitialized

```
ARCHITECTURE logic OF cmb_vari IS
BEGIN
    PROCESS(i0, i1, a)
        VARIABLE val :
                INTEGER RANGE 0 TO 1;
    BEGIN
        IF (a = '0') THEN
            val := val;
        ELSE
            val := val + 1;
        END IF;

        CASE val IS
            WHEN 0 =>
                q <= i0;
            WHEN OTHERS =>
                q <= i1;
        END CASE;
    END PROCESS;
END ARCHITECTURE logic;
```

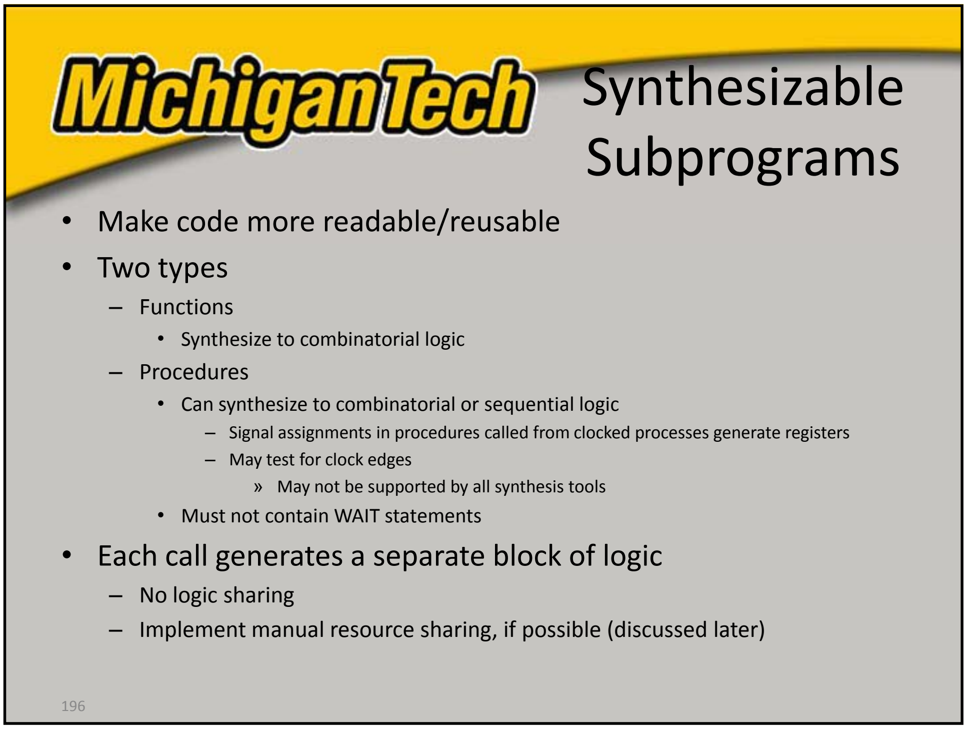
Variable used without initialization



Assign Initial Value to Variable

```
ARCHITECTURE logic OF cmb_vari IS
BEGIN
    PROCESS(i0, i1, a)
        VARIABLE val :
                INTEGER RANGE 0 TO 1;
    BEGIN
        val := 0;
        IF (a = '0') THEN
            val := val;
        ELSE
            val := val + 1;
        END IF;

        CASE val IS
            WHEN 0 =>
                q <= i0;
            WHEN OTHERS =>
                q <= i1;
        END CASE;
    END PROCESS;
END ARCHITECTURE logic;
```

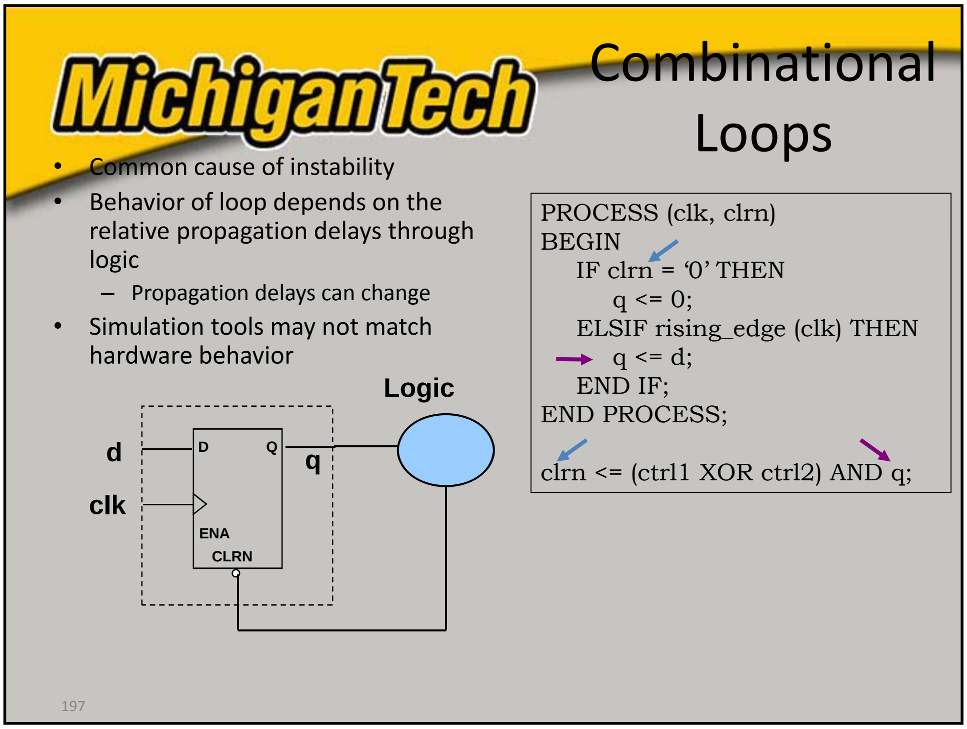
Assign initial value or signal to ***variable***



Synthesizable Subprograms

- Make code more readable/reusable
- Two types
  - Functions
    - Synthesize to combinatorial logic
  - Procedures
    - Can synthesize to combinatorial or sequential logic
      - Signal assignments in procedures called from clocked processes generate registers
      - May test for clock edges
        - » May not be supported by all synthesis tools
    - Must not contain WAIT statements
- Each call generates a separate block of logic
  - No logic sharing
  - Implement manual resource sharing, if possible (discussed later)

196

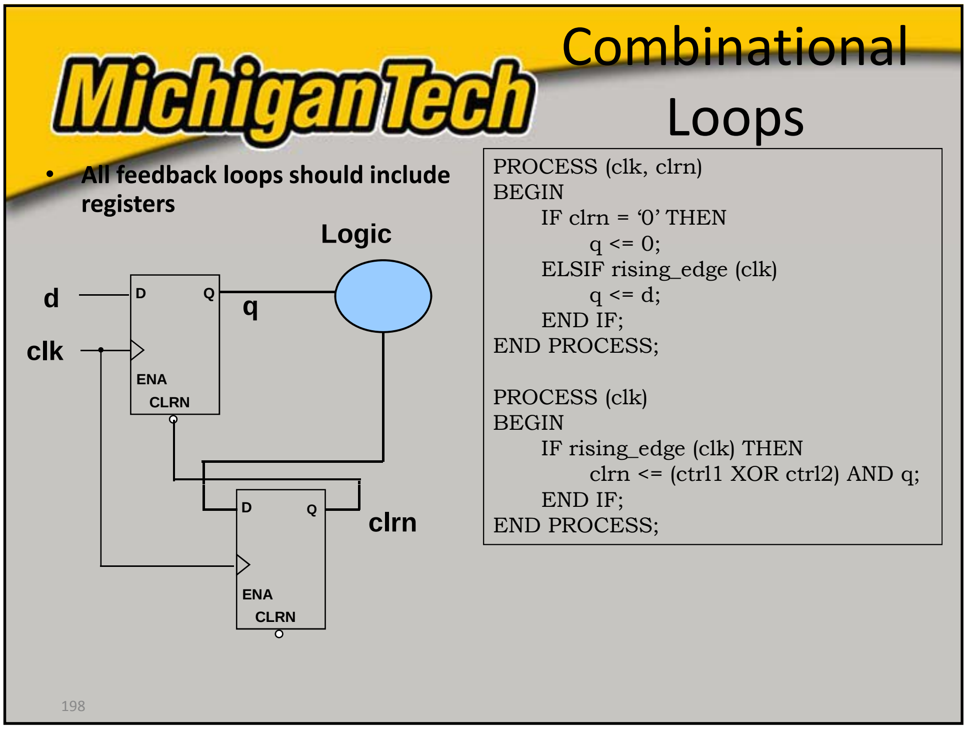


Combinational Loops

- Common cause of instability
- Behavior of loop depends on the relative propagation delays through logic
  - Propagation delays can change
- Simulation tools may not match hardware behavior

```
PROCESS (clk, clrn)
BEGIN
   IF clrn = '0' THEN
      q <= 0;
   ELSIF rising_edge (clk) THEN
      q <= d;
   END IF;
END PROCESS;

clrn <= (ctrl1 XOR ctrl2) AND q;
```

197



Combinational Loops

- **All feedback loops should include registers**

```
PROCESS (clk, clrn)
BEGIN
      IF clrn = '0' THEN
            q <= 0;
      ELSIF rising_edge (clk)
            q <= d;
      END IF;
END PROCESS;

PROCESS (clk)
BEGIN
      IF rising_edge (clk) THEN
            clrn <= (ctrl1 XOR ctrl2) AND q;
      END IF;
END PROCESS;
```

198

State Machine Coding

- Enumerated data type is used to define the different states in the state machine
  - Using constants for states may not be recognized as state machine

```
TYPE state_type IS (idle, fill, heat_w, wash, drain);
```

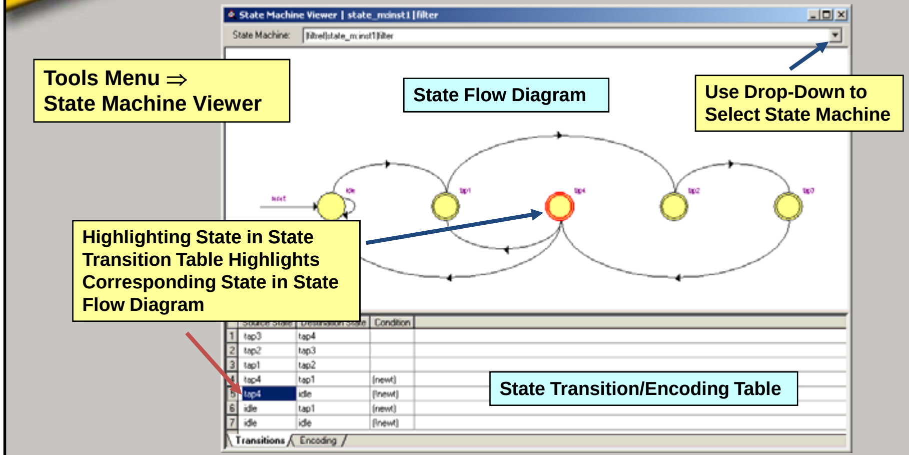
- One or two signals assigned to the name of the state-variable :

```
SIGNAL  current_state, next_state : state_type;
```

- Use CASE statement to do the next-state logic, instead of IF-THEN statement
  - Synthesis tools recognize CASE statements for implementing state machines
- Use CASE or IF-THEN-ELSE for output logic

Quartus II Software State Machine Viewer

- Use to verify correct coding of state machine



State Machine Viewer | state_m:inst1|filter
State Machine:
|filter|state_m:inst1|filter
reset
idle
tap1
tap4
tap2
tap3
Transitions
Encoding

	Source State	Destination State	Condition
1	tap3	tap4	
2	tap2	tap3	
3	tap1	tap2	
4	tap4	tap1	(newt)
5	tap4	idle	(!newt)
6	idle	tap1	(newt)
7	idle	idle	(!newt)

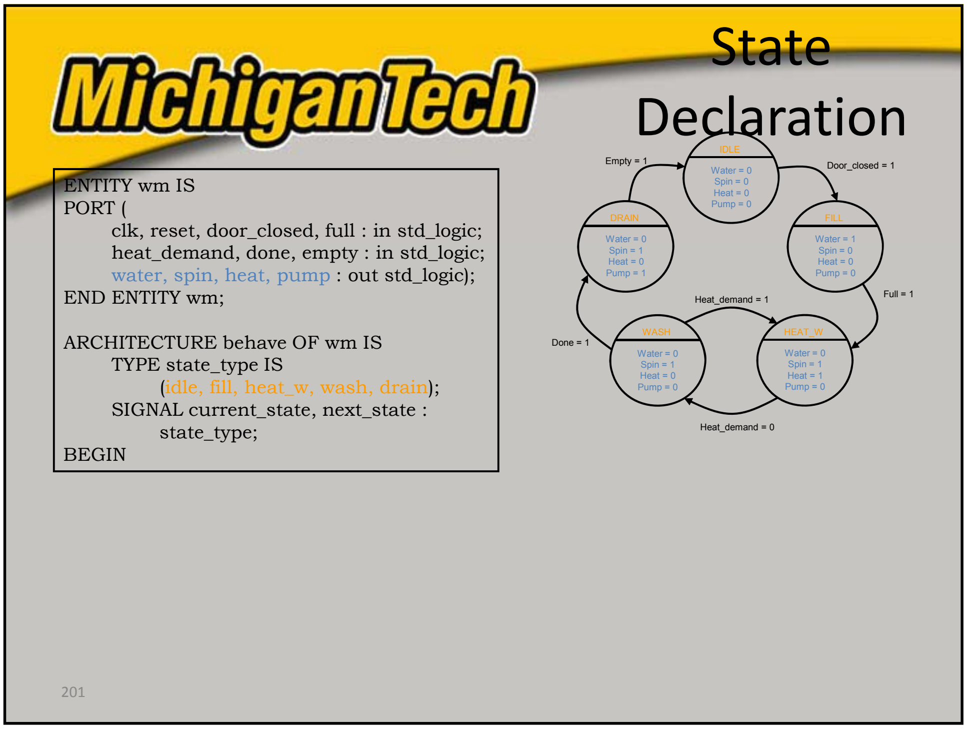
Tools Menu ⇒ State Machine Viewer

State Flow Diagram

Use Drop-Down to Select State Machine

Highlighting State in State Transition Table Highlights Corresponding State in State Flow Diagram

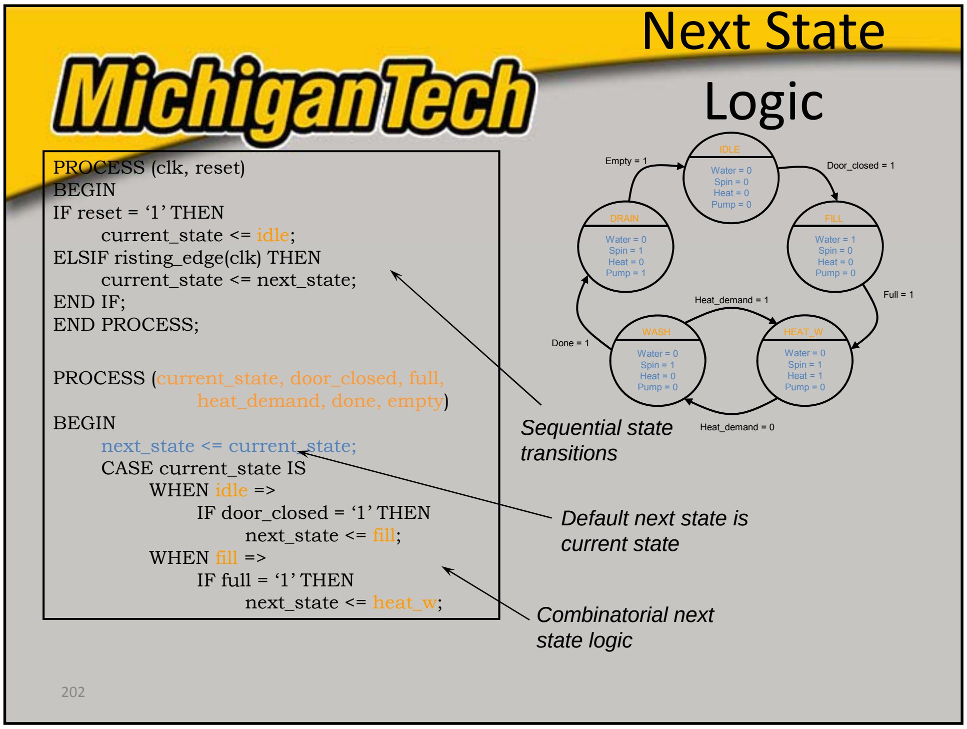
State Transition/Encoding Table



State Declaration

```
ENTITY wm IS
PORT (
        clk, reset, door_closed, full : in std_logic;
        heat_demand, done, empty : in std_logic;
        water, spin, heat, pump : out std_logic);
END ENTITY wm;

ARCHITECTURE behave OF wm IS
        TYPE state_type IS
                (idle, fill, heat_w, wash, drain);
        SIGNAL current_state, next_state :
                state_type;
BEGIN
```

201



Next State Logic

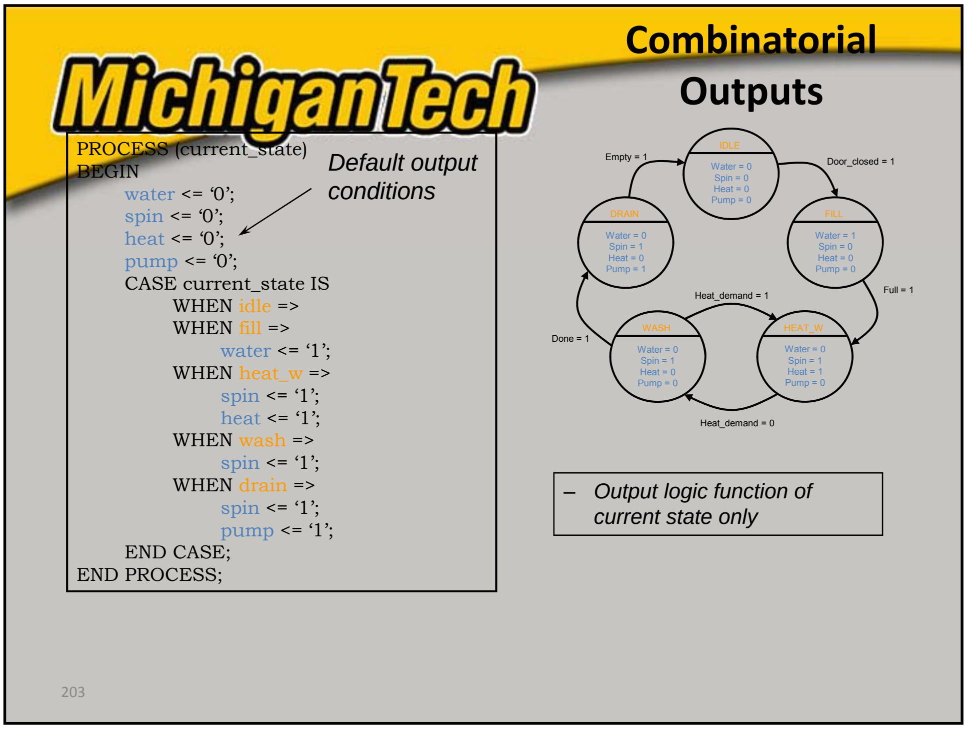
```
PROCESS (clk, reset)
BEGIN
IF reset = '1' THEN
        current_state <= idle;
ELSIF risting_edge(clk) THEN
        current_state <= next_state;
END IF;
END PROCESS;

PROCESS (current_state, door_closed, full,
                heat_demand, done, empty)
BEGIN
        next_state <= current_state;
        CASE current_state IS
                WHEN idle =>
                        IF door_closed = '1' THEN
                                next_state <= fill;
                WHEN fill =>
                        IF full = '1' THEN
                                next_state <= heat_w;
```

Sequential state transitions

Default next state is current state

Combinatorial next state logic

202



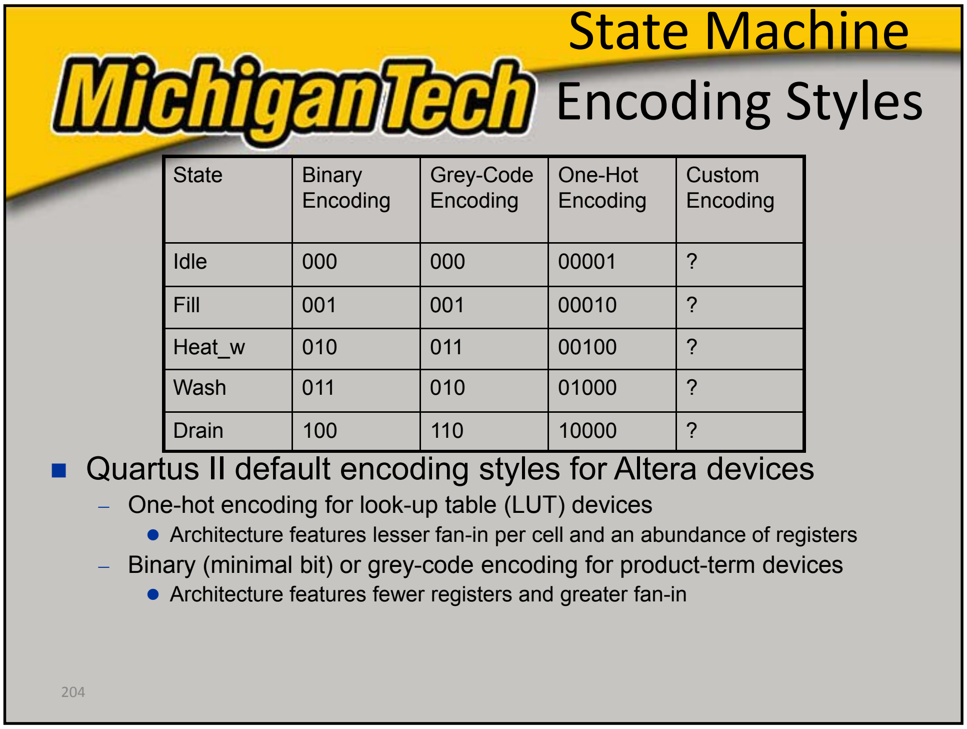
Combinatorial Outputs

```
PROCESS (current_state)
BEGIN
     water <= ‘0’;
     spin <= ‘0’;
     heat <= ‘0’;
     pump <= ‘0’;
     CASE current_state IS
          WHEN idle =>
          WHEN fill =>
               water <= ‘1’;
          WHEN heat_w =>
               spin <= ‘1’;
               heat <= ‘1’;
          WHEN wash =>
               spin <= ‘1’;
          WHEN drain =>
               spin <= ‘1’;
               pump <= ‘1’;
     END CASE;
END PROCESS;
```

Default output conditions

– *Output logic function of current state only*

203



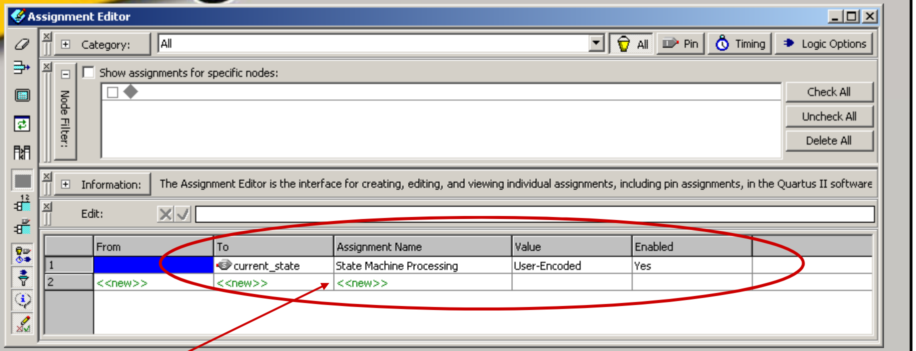
State Machine Encoding Styles

State	Binary Encoding	Grey-Code Encoding	One-Hot Encoding	Custom Encoding
Idle	000	000	00001	?
Fill	001	001	00010	?
Heat_w	010	011	00100	?
Wash	011	010	01000	?
Drain	100	110	10000	?

- Quartus II default encoding styles for Altera devices
  - One-hot encoding for look-up table (LUT) devices
    - Architecture features lesser fan-in per cell and an abundance of registers
  - Binary (minimal bit) or grey-code encoding for product-term devices
    - Architecture features fewer registers and greater fan-in

204

Quartus II Encoding Style



☐ Show assignments for specific nodes:

Information: The Assignment Editor is the interface for creating, editing, and viewing individual assignments, including pin assignments, in the Quartus II software

	From	To	Assignment Name	Value	Enabled
1		current_state	State Machine Processing	User-Encoded	Yes
2	<<new>>	<<new>>	<<new>>		

Apply Assignment to State Variable

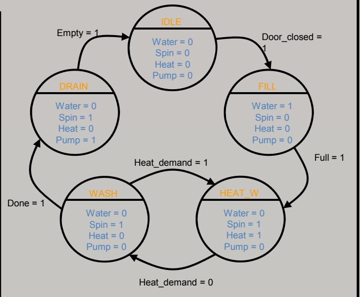
Options:
- One-Hot
- Gray
- Minimal Bits
- Sequential
- User-Encoded
- Johnson

Undefined States

- Noise and spurious events in hardware can cause state machines to enter undefined states
- If state machines do not consider undefined states, it can cause mysterious "lock-ups" in hardware
- Good engineering practice is to consider these states
- To account for undefined states
  - Explicitly code for them (manual)
  - Use "safe" synthesis constraint (automatic)

‘Safe’ Binary State Machine?

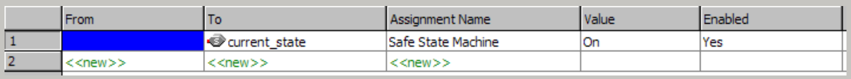
```
TYPE state_type IS
        (idle, fill, heat_w, wash, drain);
SIGNAL current_state, next_state : state_type;

PROCESS (current_state, door_closed, full, heat_demand, done, empty)
BEGIN
        next_state <= current_state;
        CASE current_state is
                WHEN idle =>
                        IF door_closed = ‘1’ THEN next_state <= fill;
                        END IF;
                WHEN fill =>
                        IF full = ‘1’ THEN next_state <= heat_w;
                        END IF;
                WHEN heat_w =>
                        IF heat_demand = ‘0’ THEN next_state <= wash;
                        END IF;
                WHEN wash =>
                        IF heat_demand = ‘1’ THEN next_state <= heat_w;
                        ELSIF done = ‘1’ THEN next_state <= drain;
                        END IF;
                WHEN drain =>
                        IF empty = ‘1’ THEN next_state <= idle;
                        END IF;
                WHEN others =>
                        next_state <= idle;
        END CASE;
END PROCESS;
```



- *This code does not consider undefined states*
- *The “when others” statement only considers other enumerated states*
- *The states “101”, “110” & “111” are not considered*

Creating “Safe” State Machines

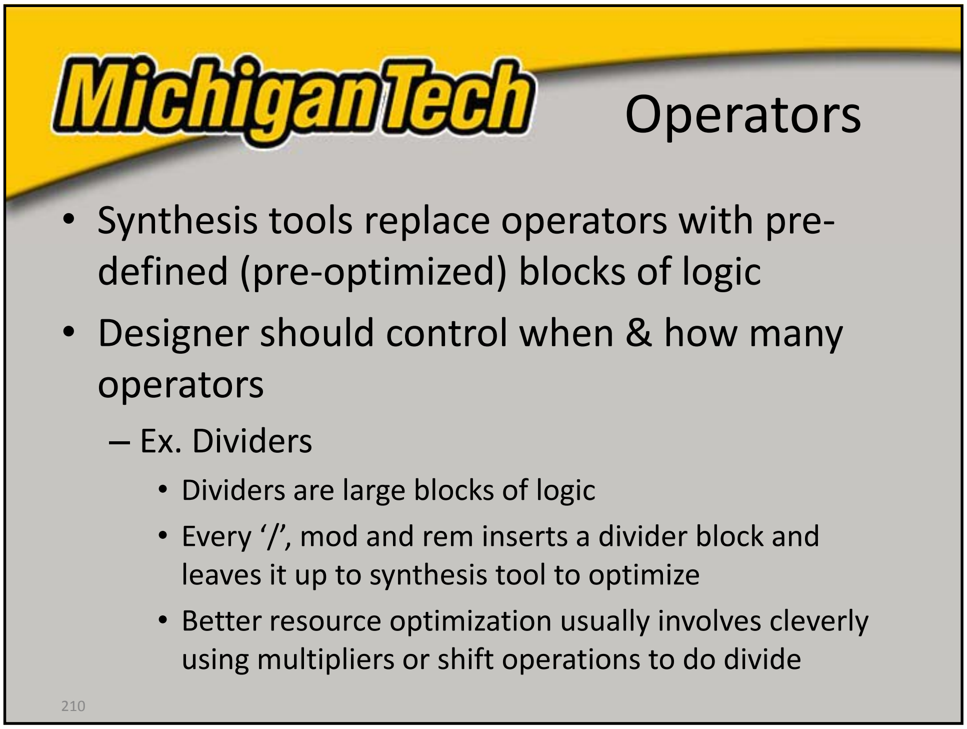
- WHEN OTHERS clause does not make state machines “safe”
  - Once state machine is recognized, synthesis tool only accounts for explicitly defined states
  - Exception: Number of states equals power of 2 **AND** binary/grey encoding enabled
- Safe state machines created using synthesis constraints
  - Quartus II software uses
    - SAFE STATE MACHINE assignment applied project-wide and to individual FSMs
    - VHDL synthesis attribute
  - May increase logic usage



	From	To	Assignment Name	Value	Enabled
1		current_state	Safe State Machine	On	Yes
2	<<new>>	<<new>>	<<new>>		

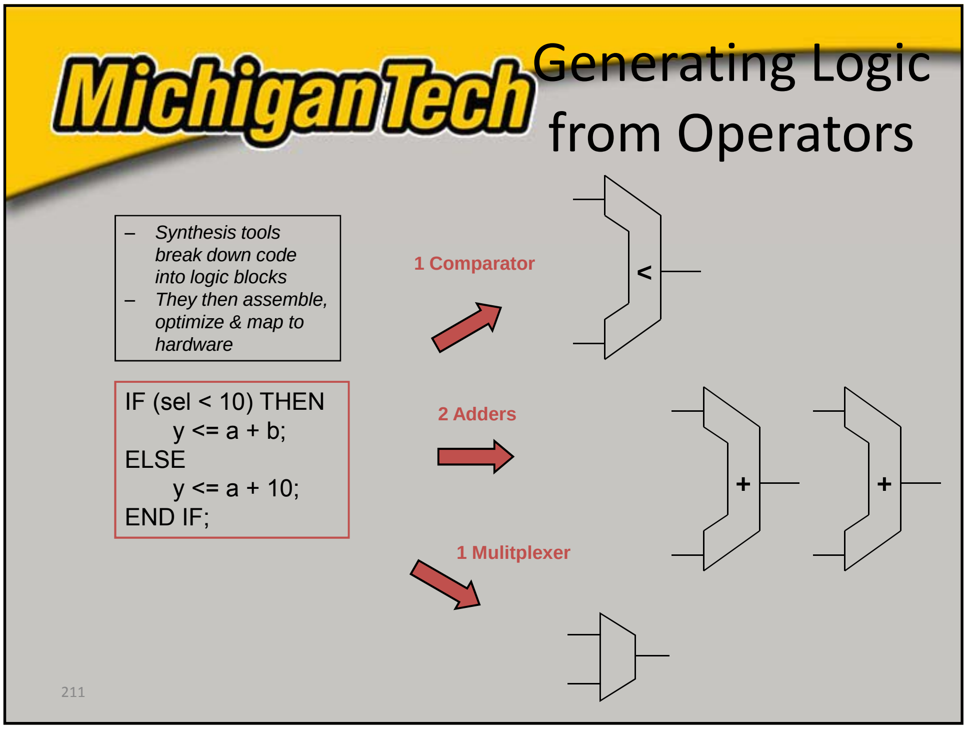
VHDL Logic Optimization & Performance

- Balancing operators
- Resource sharing
- Logic duplication
- Pipelining



Operators

- Synthesis tools replace operators with pre-defined (pre-optimized) blocks of logic
- Designer should control when & how many operators
  - Ex. Dividers
    - Dividers are large blocks of logic
    - Every '/', mod and rem inserts a divider block and leaves it up to synthesis tool to optimize
    - Better resource optimization usually involves cleverly using multipliers or shift operations to do divide

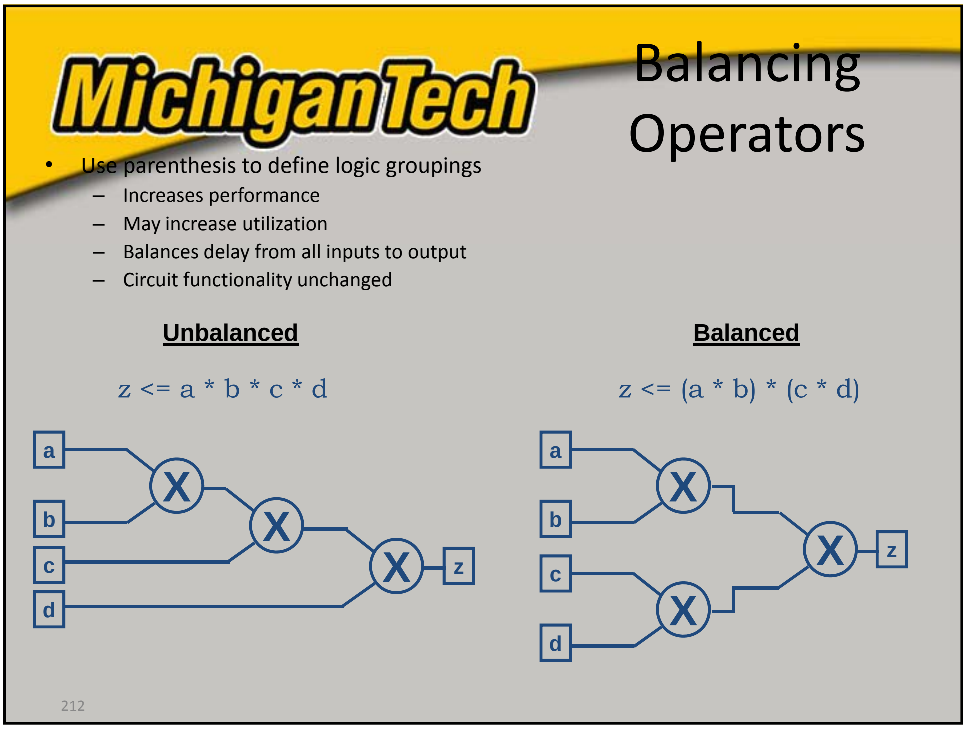


Generating Logic from Operators

- *Synthesis tools break down code into logic blocks*
- *They then assemble, optimize & map to hardware*

```
IF (sel < 10) THEN
       y <= a + b;
ELSE
       y <= a + 10;
END IF;
```

211



Balancing Operators

- Use parenthesis to define logic groupings
  - Increases performance
  - May increase utilization
  - Balances delay from all inputs to output
  - Circuit functionality unchanged

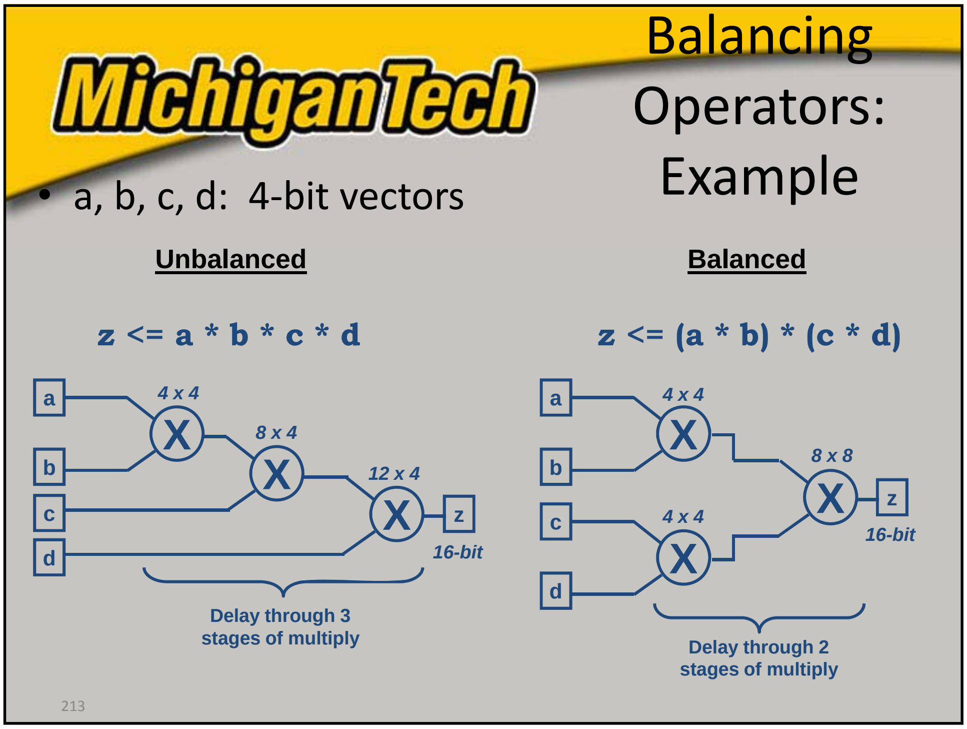
Unbalanced

z <= a * b * c * d

Balanced

z <= (a * b) * (c * d)

212



Balancing Operators: Example

- a, b, c, d: 4-bit vectors

Unbalanced

z <= a * b * c * d

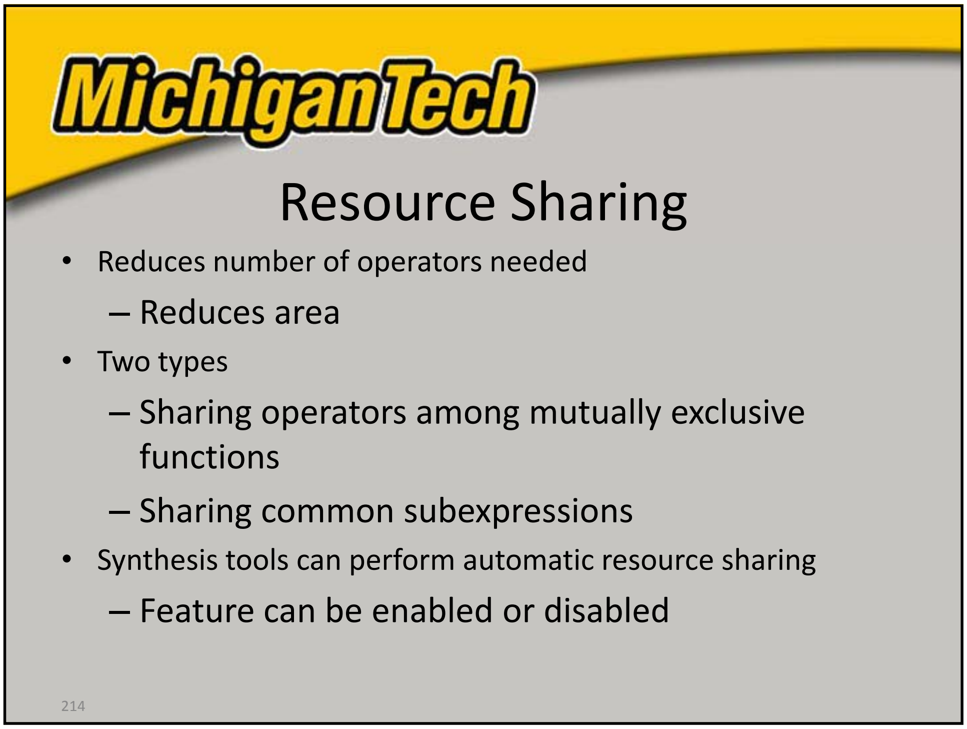
Delay through 3 stages of multiply

Balanced

z <= (a * b) * (c * d)

Delay through 2 stages of multiply

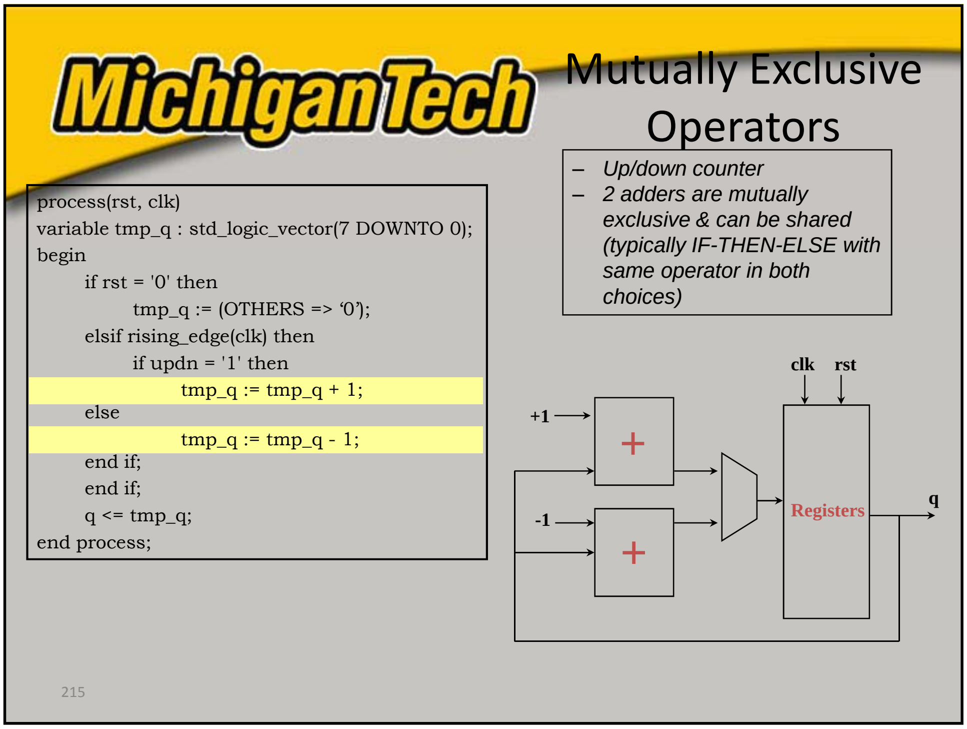
213



Resource Sharing

- Reduces number of operators needed
  - Reduces area
- Two types
  - Sharing operators among mutually exclusive functions
  - Sharing common subexpressions
- Synthesis tools can perform automatic resource sharing
  - Feature can be enabled or disabled

214

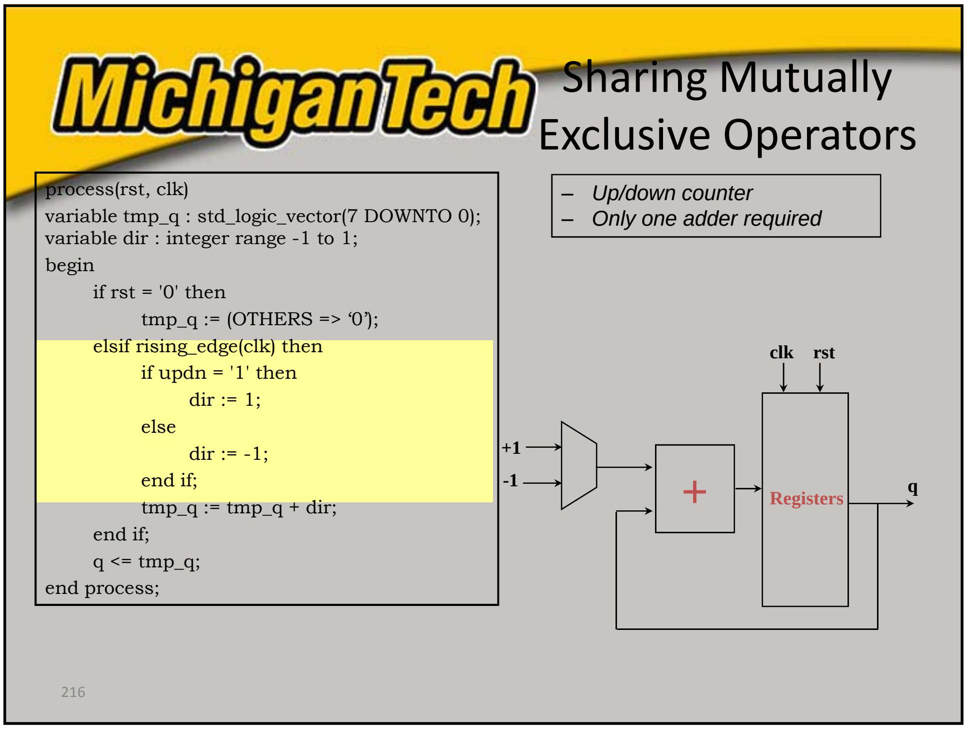


Mutually Exclusive Operators

- Up/down counter
- 2 adders are mutually exclusive & can be shared (typically IF-THEN-ELSE with same operator in both choices)

```
process(rst, clk)
variable tmp_q : std_logic_vector(7 DOWNTO 0);
begin
        if rst = '0' then
                tmp_q := (OTHERS => ‘0’);
        elsif rising_edge(clk) then
                if updn = '1' then
                        tmp_q := tmp_q + 1;
        else
                        tmp_q := tmp_q - 1;
        end if;
        end if;
        q <= tmp_q;
end process;
```

215

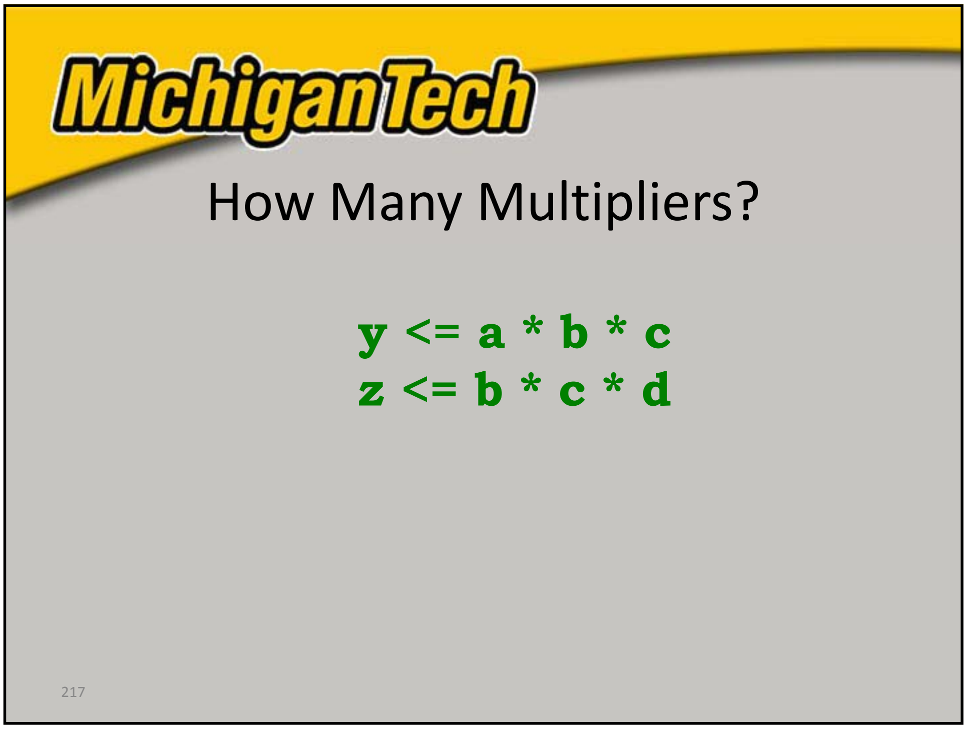


Sharing Mutually Exclusive Operators

- Up/down counter
- Only one adder required

```
process(rst, clk)
variable tmp_q : std_logic_vector(7 DOWNTO 0);
variable dir : integer range -1 to 1;
begin
        if rst = '0' then
                tmp_q := (OTHERS => ‘0’);
        elsif rising_edge(clk) then
                if updn = '1' then
                        dir := 1;
                else
                        dir := -1;
                end if;
                tmp_q := tmp_q + dir;
        end if;
        q <= tmp_q;
end process;
```

216

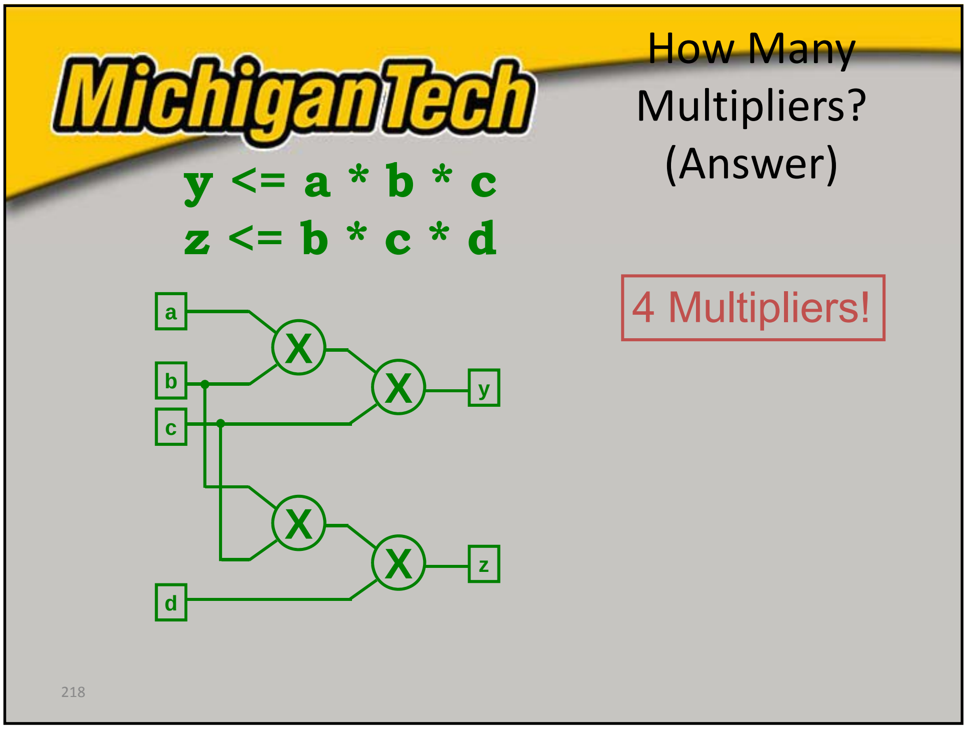


MichiganTech

How Many Multipliers?

```
y <= a * b * c
z <= b * c * d
```

217



MichiganTech

How Many Multipliers? (Answer)

```
y <= a * b * c
z <= b * c * d
```

4 Multipliers!

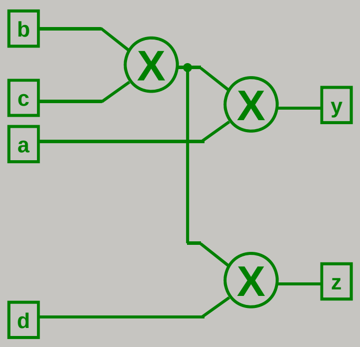
218

How Many Multipliers Again?

y <= a * (b * c)
z <= (b * c) * d

How Many Multipliers Again? (Answer)

y <= a * (b * c)
z <= (b * c) * d

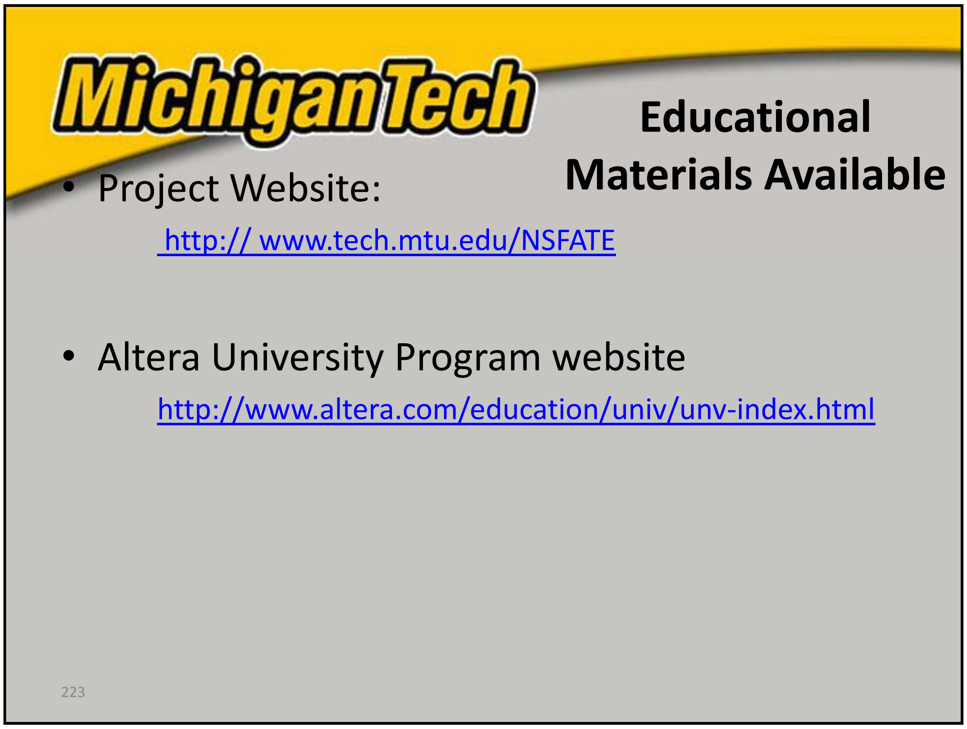


3 Multipliers!

- *This is called sharing common subexpressions*
- *Some synthesis tools do this automatically, but some don't!*
- *Parentheses guide synthesis tools*
- *If (b*c) is used repeatedly, assign to temporary signal*

Please go to Lab4
Advanced VHDL
in the Manual

Please go to
Final Project
in the Manual



Educational Materials Available

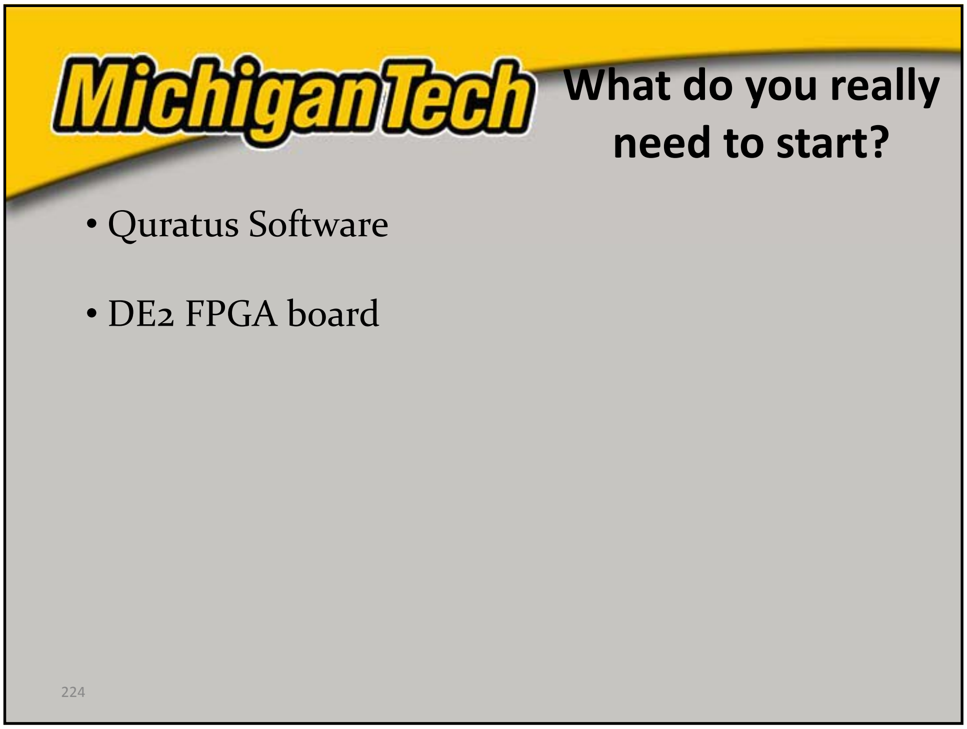
- Project Website:

  [http:// www.tech.mtu.edu/NSFATE](http://www.tech.mtu.edu/NSFATE)

- Altera University Program website

  http://www.altera.com/education/univ/unv-index.html

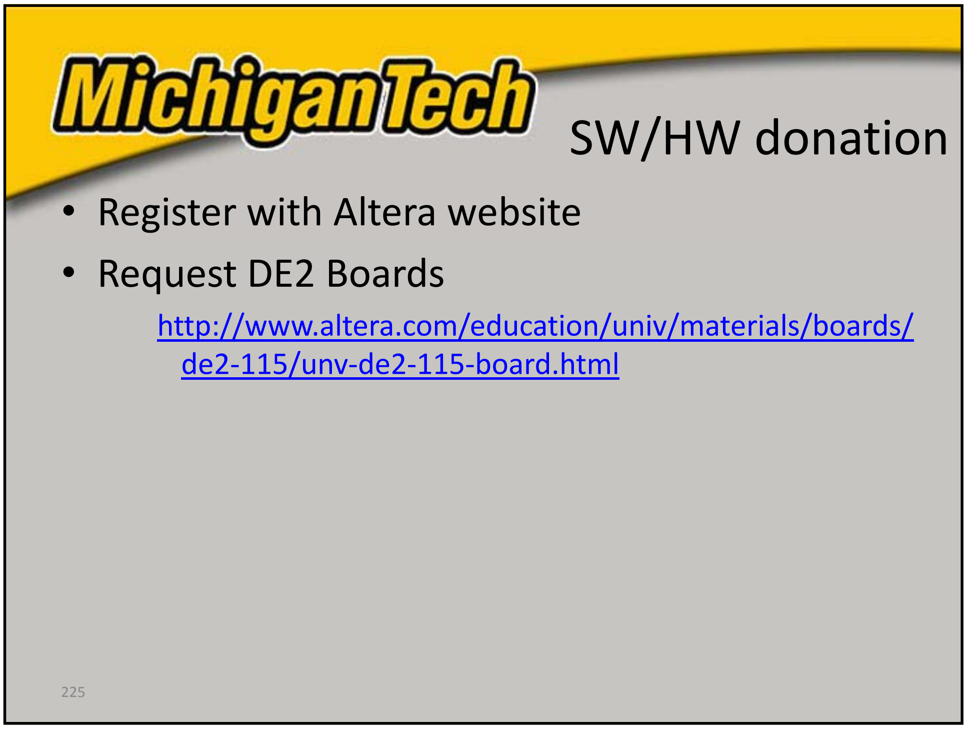
223



What do you really need to start?

- Quratus Software
- DE2 FPGA board

224

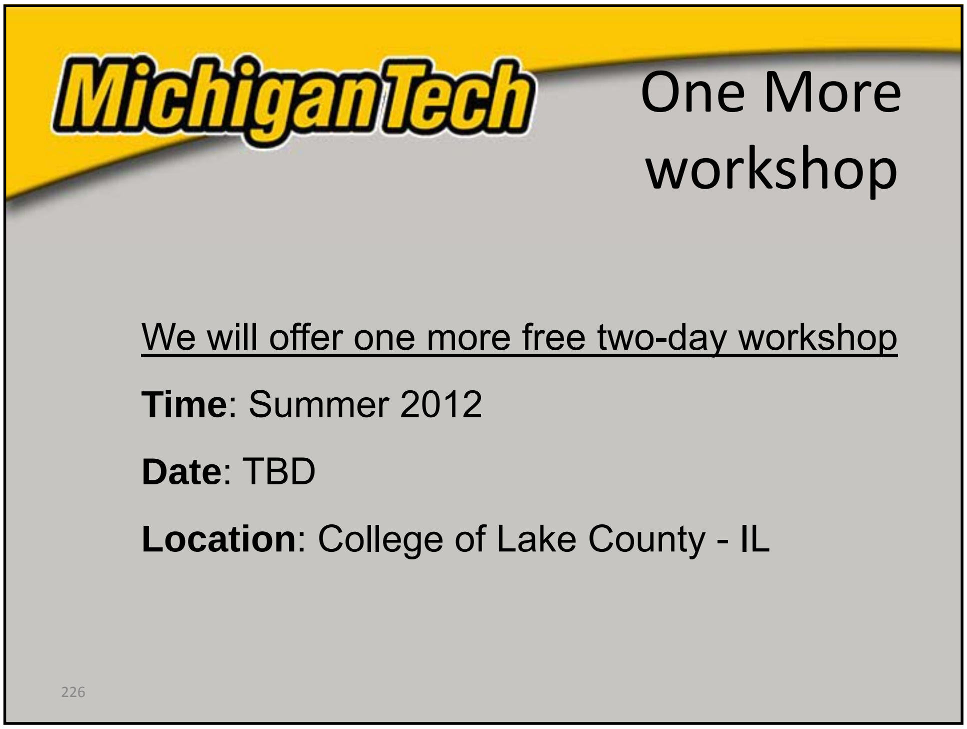


MichiganTech

SW/HW donation

- Register with Altera website
- Request DE2 Boards

  http://www.altera.com/education/univ/materials/boards/de2-115/unv-de2-115-board.html



MichiganTech

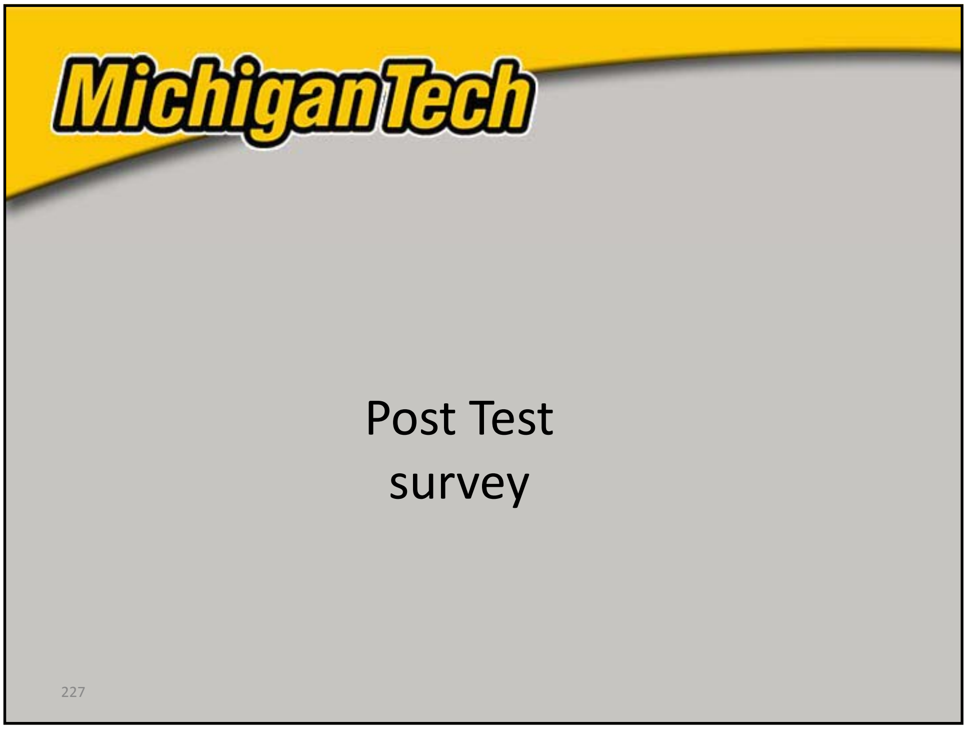
One More workshop

We will offer one more free two-day workshop

Time: Summer 2012

Date: TBD

Location: College of Lake County - IL



MichiganTech

Post Test
survey

227

MichiganTech

Thank You
?

Thanks to
National Science Foundation
Advanced Technological Education

NSF

228

228

Lab Manual

for VHDL and FPGA Design Workshop

Software Requirements to complete all exercises
Quartus II software version 9.1 or newer

Link to the Quartus II Handbook:
http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf

Link to the project web page to download the related lab files:
http://www.tech.mtu.edu/NSFATE/

Lab 1

Introduction to Quartus II

Lab 1 – Introduction to Quartus II

This lab is designed to familiarize you with using many of the common aspects of the Quartus II software through a complete design phase. You will create a new project, create a new vhdl file, use the MegaWizard Plug-In Manager, compile the design, plan and manage I/O assignments, apply timing analysis using the TimeQuest Timing Analyzer, write Synopsys Design Contraint (SDC) files, and program a design onto the Altera DE2 Development Board.

Task 1: Create a New Project

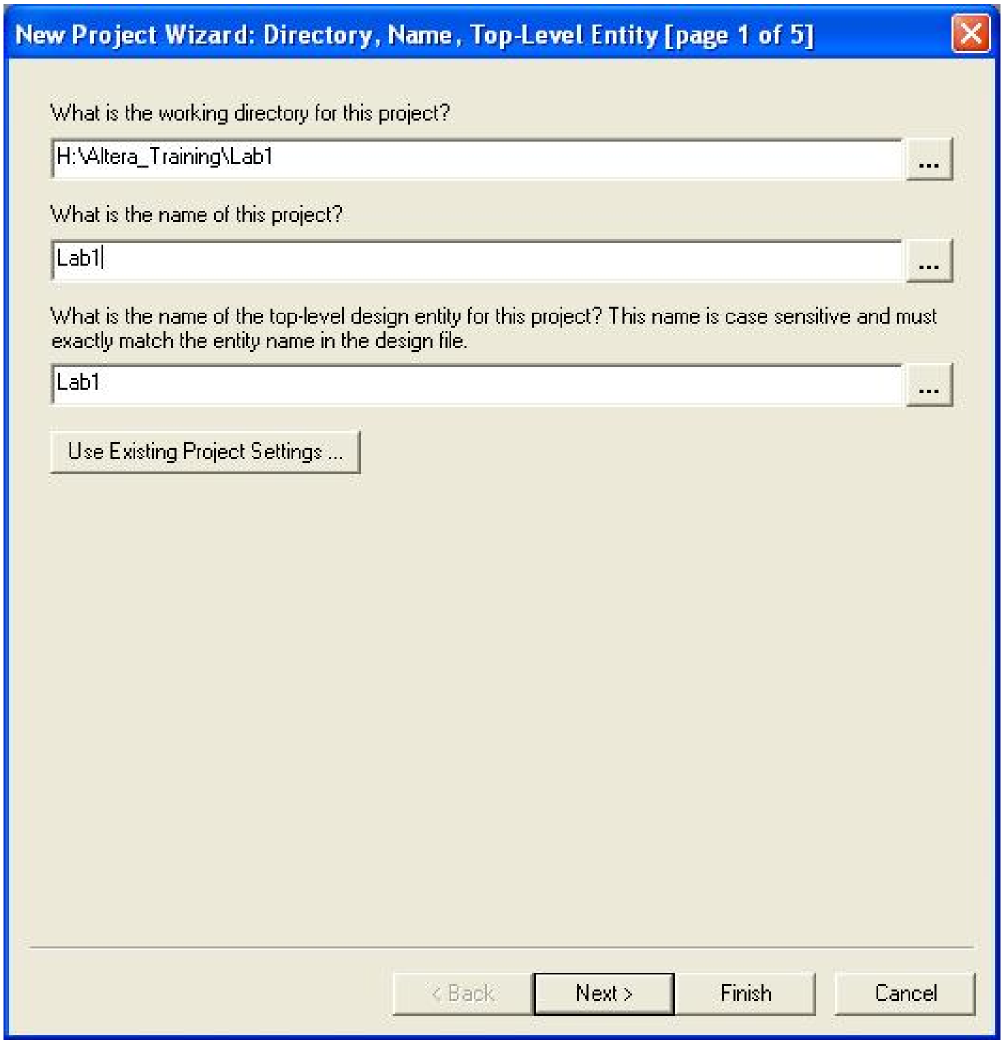
1. **Start the Quartus II software.**
   From the Windows Start Menu, select:
   All Programs → Other Apps → Altera → Quartus II 9.1 → Quartus II 9.1 (32-Bit)

2. **Start the New Project Wizard.**
   If the opening splash screen is displayed, select: **Create a New Project (New Project Wizard)**,
   otherwise from the Quartus II Menu Bar select: **File → New Project Wizard**.

3. **Select the Working Directory and Project Name.**

Working Directory	H:\Altera_Training\Lab1
Project Name	Lab1
Top-Level De ign Entity	Lab1

Click **Next** to advance to page 2 of the New Project Wizard.
*Note: A window may pop up stating that the chosen working directory does not exist. Click **Yes** to create it.*



New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

H:\Altera_Training\Lab1

What is the name of this project?

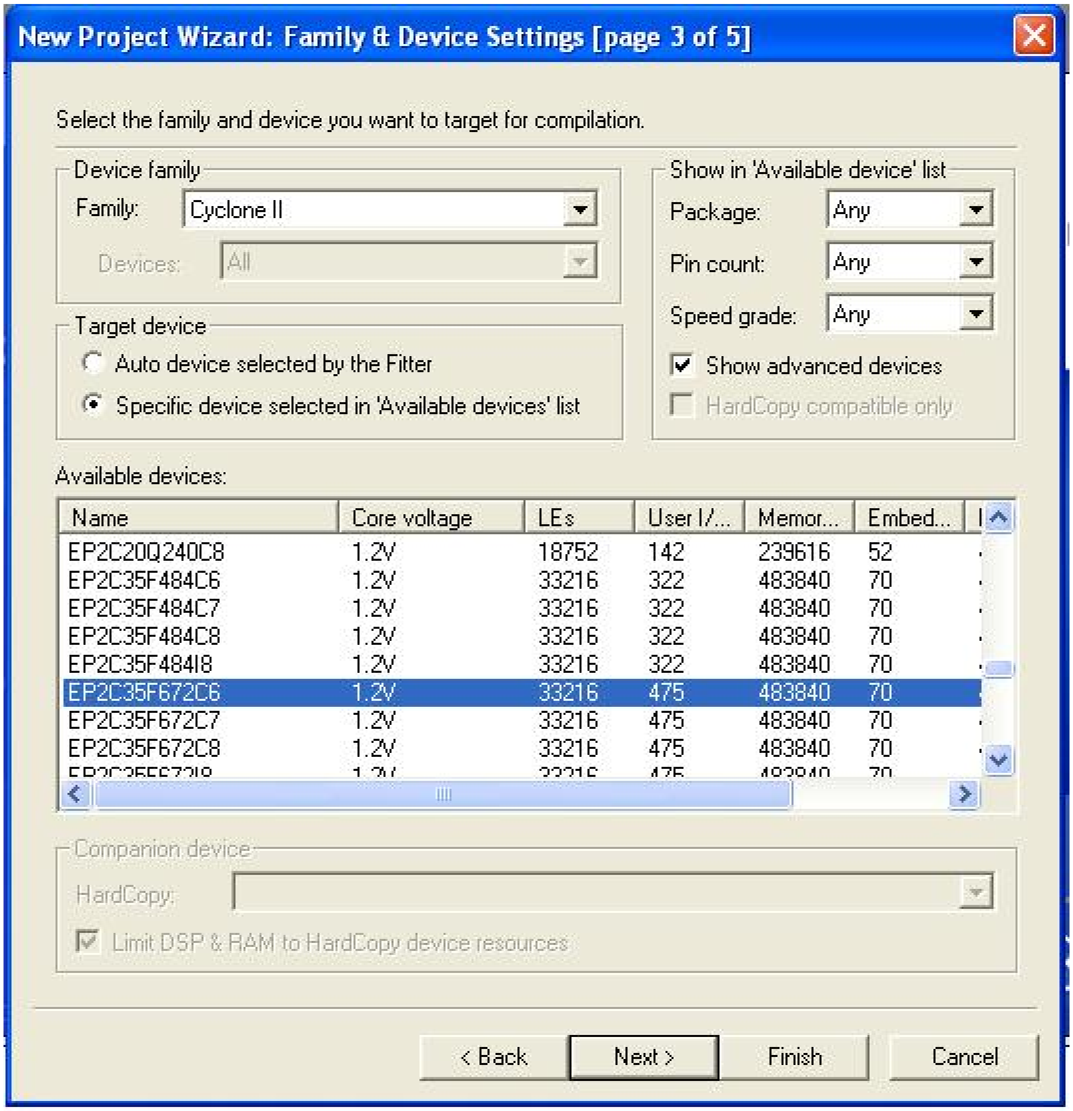
Lab1

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

Lab1

4. Click **Next** again as we will not be adding any preexisting design files at this time.

5. **Select the family and Device Settings.**
   From the pull-down menu labeled **Family**, select **Cyclone II**.
   In the list of available devices, select **EPC235F672C6.**
   Click **Next.**



New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family
Family: Cyclone II
Devices: All

Show in 'Available device' list
Package: Any
Pin count: Any
Speed grade: Any
☑ Show advanced devices
□ HardCopy compatible only

Target device
○ Auto device selected by the Fitter
● Specific device selected in 'Available devices' list

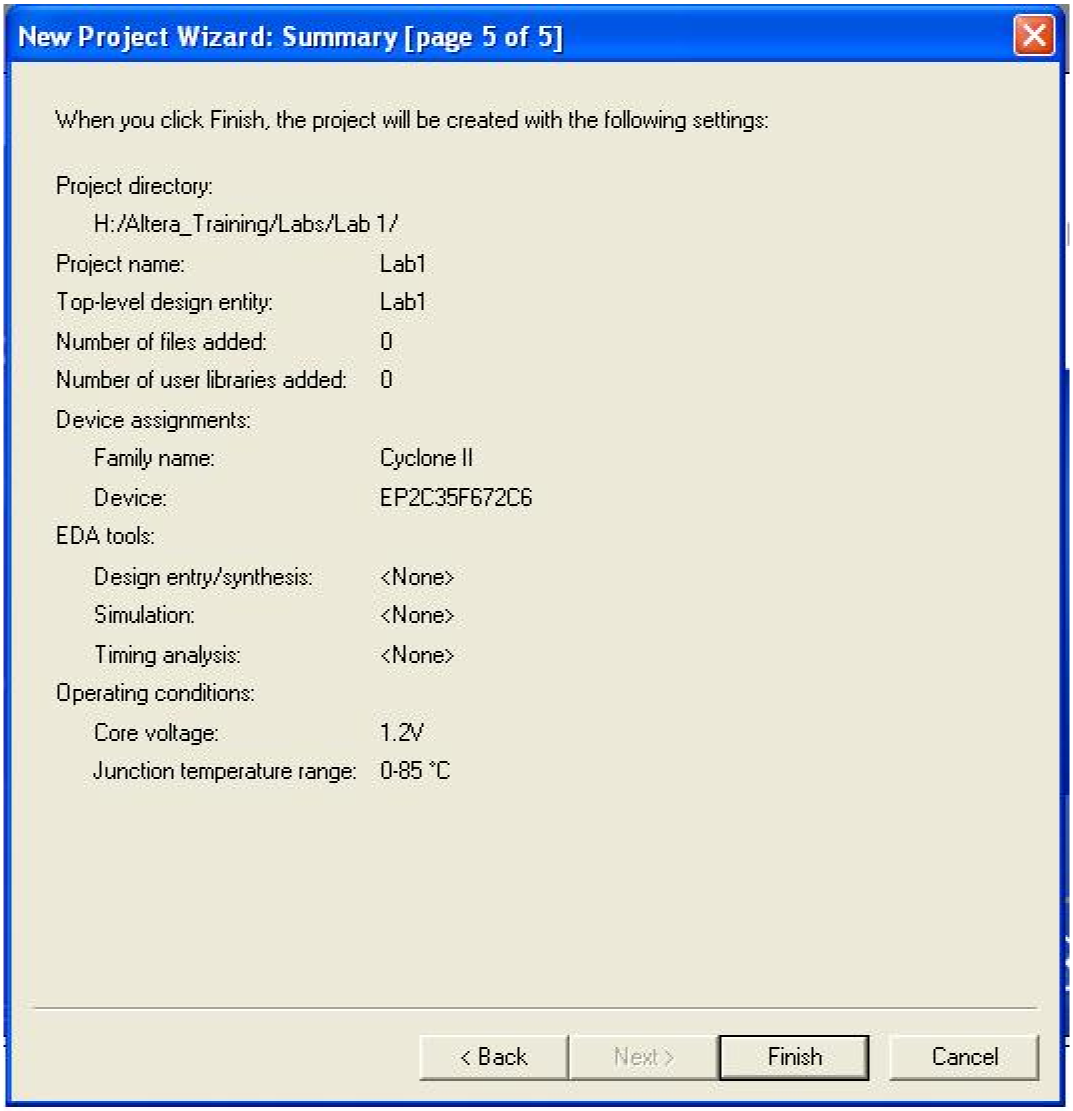
Available devices:

Name	Core voltage	LEs	User I/...	Memor...	Embed...
EP2C20Q240C8	1.2V	18752	142	239616	52
EP2C35F484C6	1.2V	33216	322	483840	70
EP2C35F484C7	1.2V	33216	322	483840	70
EP2C35F484C8	1.2V	33216	322	483840	70
EP2C35F484I8	1.2V	33216	322	483840	70
EP2C35F672C6	1.2V	33216	475	483840	70
EP2C35F672C7	1.2V	33216	475	483840	70
EP2C35F672C8	1.2V	33216	475	483840	70
EP2C35F672I8	1.2V	33216	475	483840	70

Companion device
HardCopy:
☑ Limit DSP & RAM to HardCopy device resources

6. Click **Next** again as we will not be using any third party EDA tools.

7. Click **Finish** to complete the New Project Wizard.

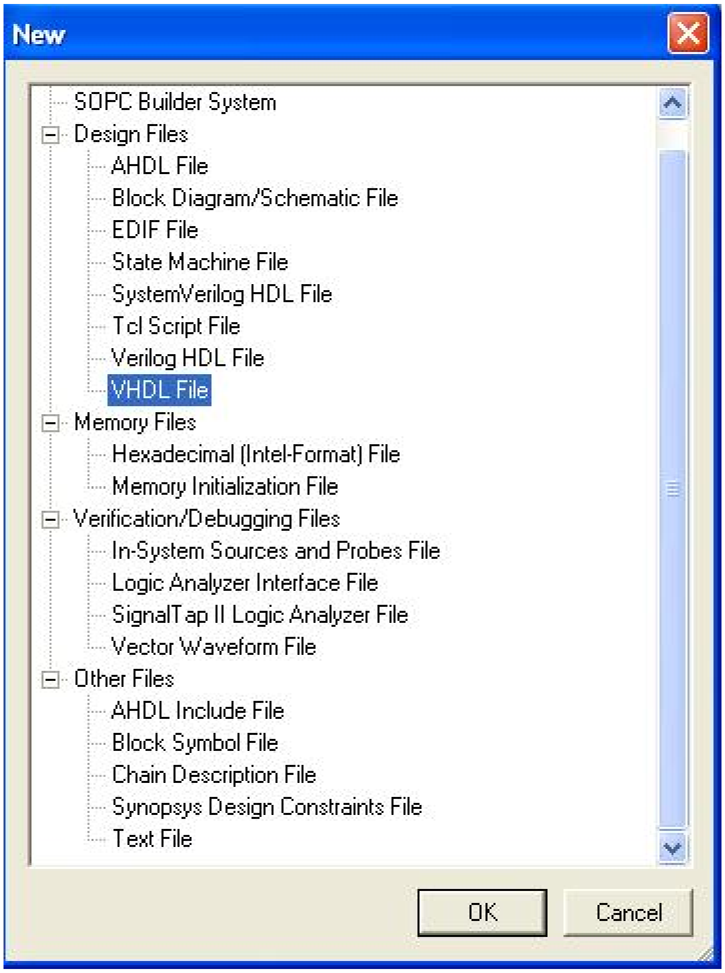


When you click Finish, the project will be created with the following settings:

Project directory:	
H:/Altera_Training/Labs/Lab 1/	
Project name:	Lab1
Top-level design entity:	Lab1
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone II
Device:	EP2C35F672C6
EDA tools:	
Design entry/synthesis:	<None>
Simulation:	<None>
Timing analysis:	<None>
Operating conditions:	
Core voltage:	1.2V
Junction temperature range:	0-85 °C

Task 2: Create, Add, and Compile Design Files

1. **Create a new Design File.**
   Select: **File → New** from the Menu Bar.
   Select: **VHDL File** from the **Design Files** list and click **OK.**

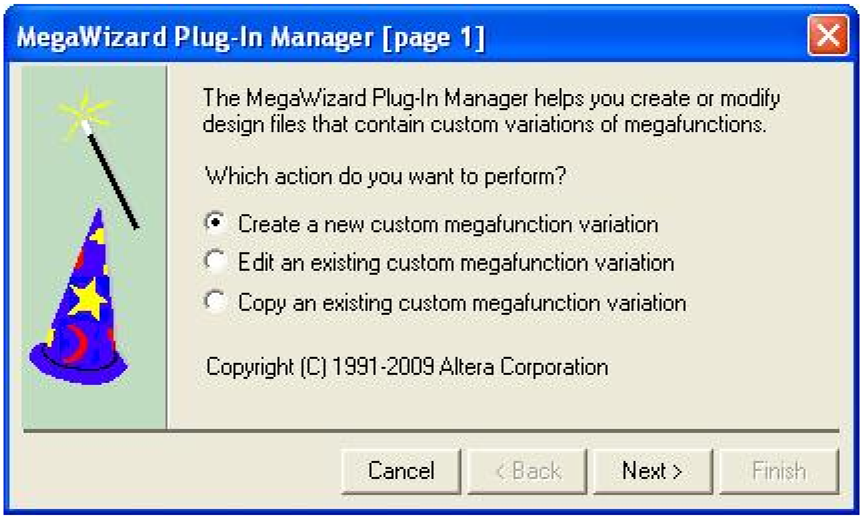


2. Copy and paste the following code into your new VHDL file, then save it by selecting **File → Save**. Keep the default file name and click **Save** in the **Save As** dialog box.

Lab 1 – Introduction to Quartus II – VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Lab1 IS
      PORT (clk: IN STD_LOGIC;
            input0, input1, input2: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
            sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            output: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END Lab1;

ARCHITECTURE Structure OF Lab1 IS

COMPONENT Mux3x1 IS
      PORT (clock: IN STD_LOGIC;
            data0x, data1x, data2x: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
            sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            result: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END COMPONENT;

BEGIN

      MuxOut: Mux3x1 PORT MAP (clk, input0, input1, input2, sel, output);

END Structure;
```

3. **Create an 4-Bit 3x1 Multiplexer using the MegaWizard® Plug-in Manager**
Select: **Tools → MegaWizard Plug-In Manager**
Select the radio button for **Create a new custom megafunction variation** and click **Next**.



MegaWizard Plug-In Manager [page 1]

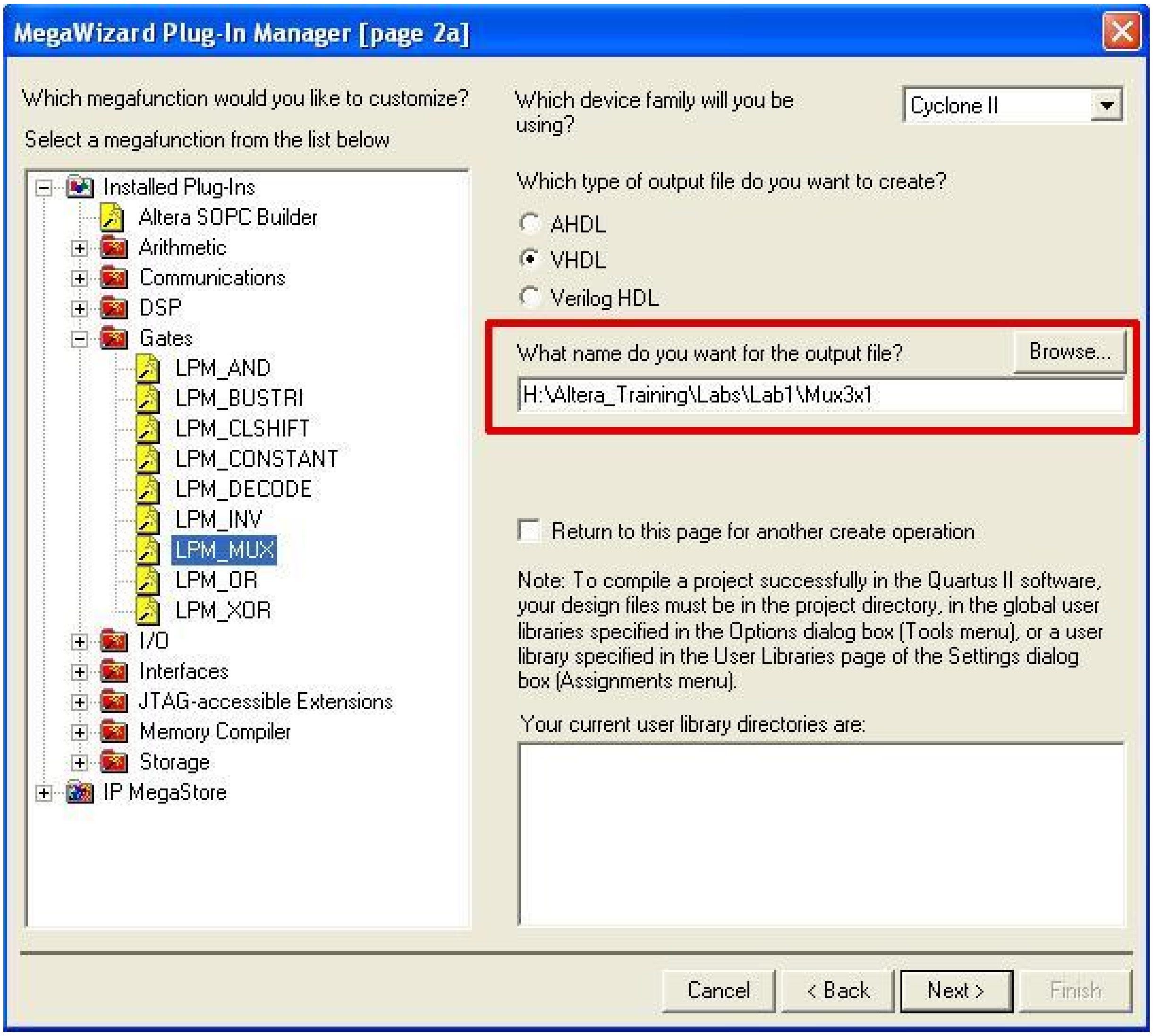
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

● Create a new custom megafunction variation
○ Edit an existing custom megafunction variation
○ Copy an existing custom megafunction variation

Copyright (C) 1991-2009 Altera Corporation

4. From the **Gates** menu, select the **LPM_MUX** subitem and name the file **Mux3x1** and click **Next**.



MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?

Select a megafunction from the list below

Which device family will you be using? Cyclone II

Which type of output file do you want to create?

○ AHDL

● VHDL

○ Verilog HDL

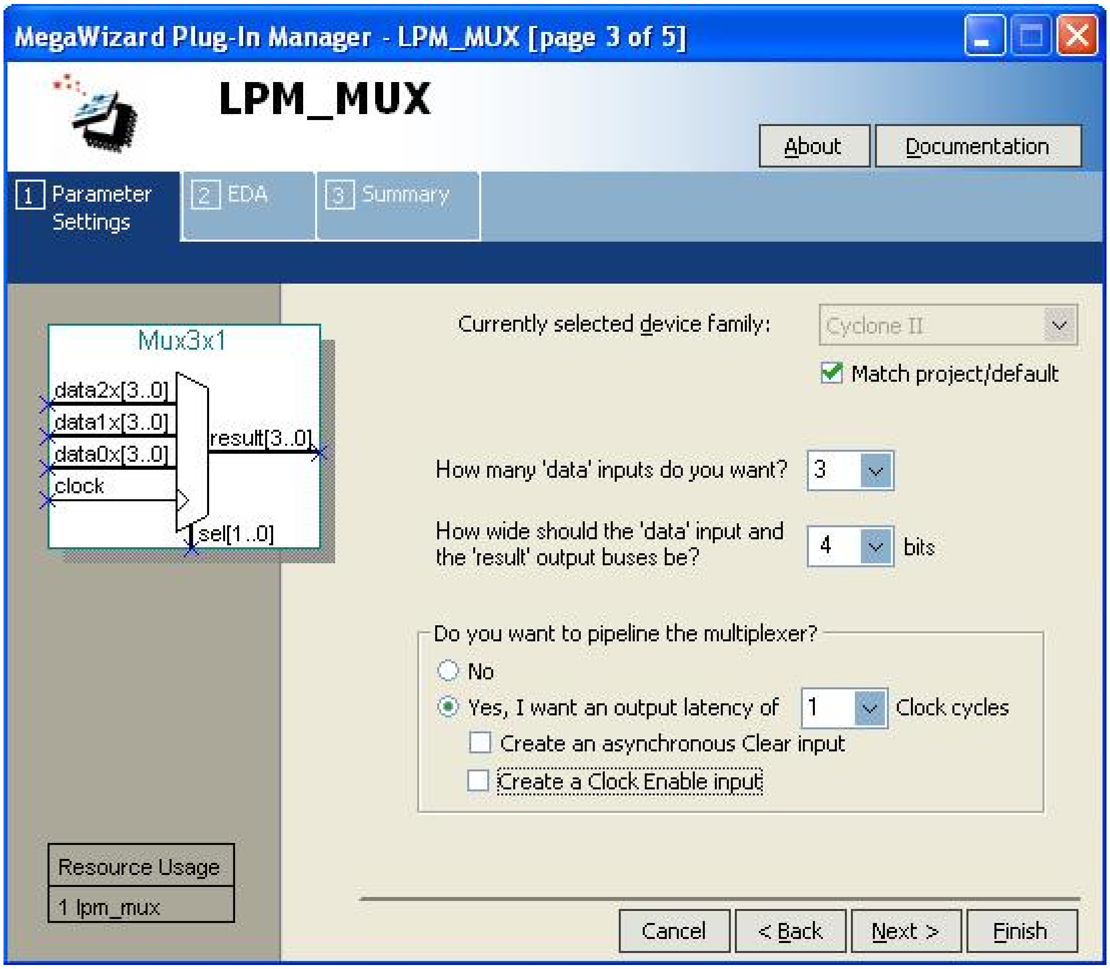
What name do you want for the output file? H:\Altera_Training\Labs\Lab1\Mux3x1

□ Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

5. Select **3** for the number of data inputs from the drop-down box.
   Select **4** bits for the width of the data input and result output buses from the drop-down box.
   Under **Do you want to pipeline the multiplexer?** Select **Yes** and set the output latency to **1** clock cycle.
   Click **Next**.



MegaWizard Plug-In Manager - LPM_MUX [page 3 of 5]

LPM_MUX

Currently selected device family: Cyclone II

☑ Match project/default

How many 'data' inputs do you want? 3

How wide should the 'data' input and the 'result' output buses be? 4 bits

Do you want to pipeline the multiplexer?

○ No

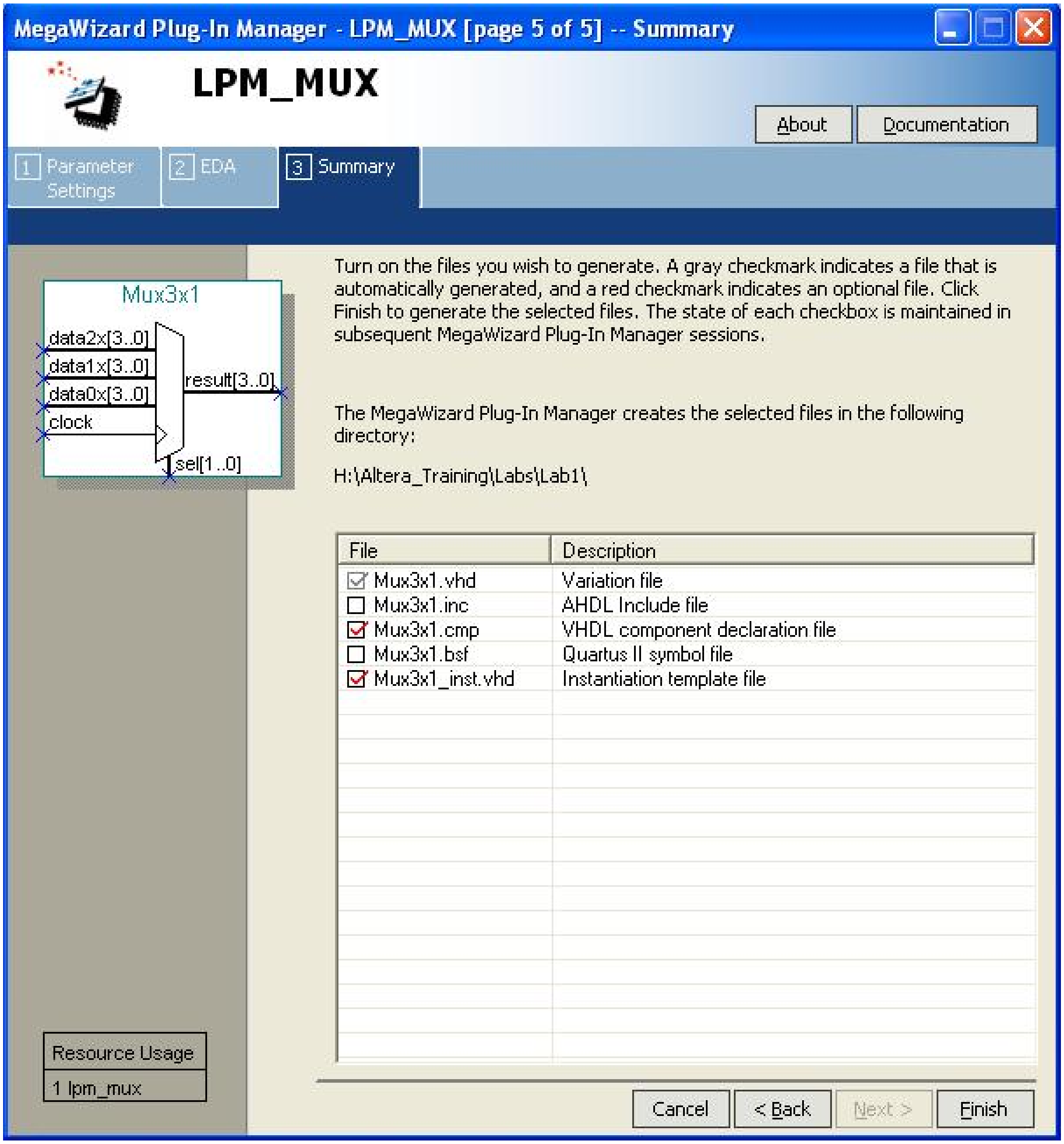
● Yes, I want an output latency of 1 Clock cycles

□ Create an asynchronous Clear input

□ Create a Clock Enable input

6. On page 4 of the MegaWizard Plug-In Manager, click **Next**.

7. Verify that **Mux3x1.vhd**, **Mux3x1.cmp**, and **Mux3x1_inst.vhd** are selected to be created and click **Finish**.
   A dialog box may appear about the creation of a Quartus II Ip File. If so, simply click **Yes**.



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a red checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

H:\Altera_Training\Labs\Lab1\

File	Description
☑ Mux3x1.vhd	Variation file
□ Mux3x1.inc	AHDL Include file
☑ Mux3x1.cmp	VHDL component declaration file
□ Mux3x1.bsf	Quartus II symbol file
☑ Mux3x1_inst.vhd	Instantiation template file

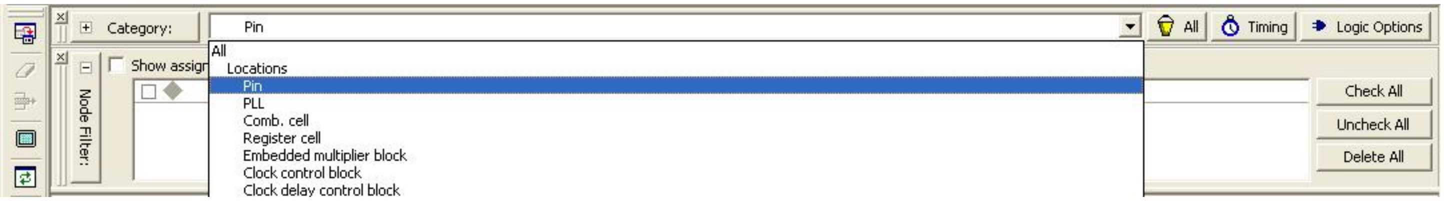
8. **Compile the Design.**
   To perform a full compilation select: **Processing → Start Compilation**

   Alternatively, select the Start Compilation button on the toolbar ▶
   You may ignore any warnings that may appear at this time.

Task 3: Assign Pins

1. From the Menu Bar select: **Assignments → Assignment Editor**

2. From the drop-down menu at the top, for **Category**, select **Pin**.



3. Double-click **<<new>>** in the **To** column and select **input0(0)**.
   Double-click the adjacent cell under **Location** and type **PIN_AF14**.
   Continue to assign the pins as seen in the table below.

	To	Location	DE2 Board Description
1	input0(0)	PIN_AF14	SW4
2	input0(1)	PIN_AD13	SW5
3	input0(2)	PIN_AC13	SW6
4	input0(3)	PIN_C13	SW7
5	input1(0)	PIN_A13	SW9
6	iInput1(1)	PIN_N1	SW10
7	input1(2)	PIN_P1	SW11
8	input1(3)	PIN_P2	SW12
9	input2(0)	PIN_U3	SW14
10	input2(1)	PIN_U4	SW15
11	input2(2)	PIN_V1	SW16
12	input2(3)	PIN_V2	SW17
13	sel(0)	PIN_N25	SW0
14	sel(1)	PIN_N26	SW1
15	output(0)	PIN_AE22	LEDG0
16	output(1)	PIN_AF22	LEDG1
17	output(2)	PIN_W19	LEDG2
18	output(3)	PIN_V18	LEDG3
19	Clk	PIN_N2	50MHz On Board Clock

Note: A complete list of pin assignments for the DE2 Development Board can be found here:
http://www.terasic.com.tw/attachment/archive/30/DE2_Pin_Table.pdf

4. Save the pin assignments by selecting **File → Save** from the Menu Bar, or by clicking the Save button 💾 on the toolbar.

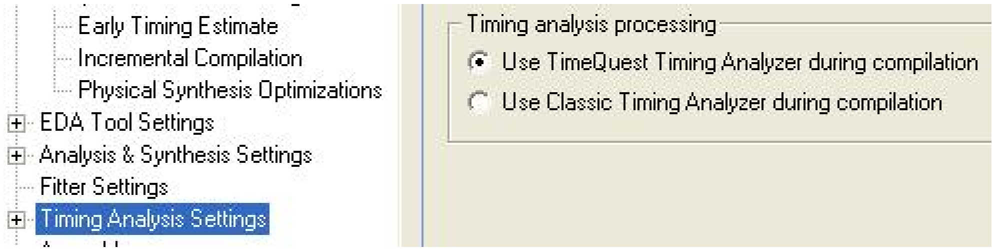
5. **View the Pin Assignments**
   Select **Assignments → Pin Planner** to open the Pin Planner window.
   Here you can view which pins have been assigned.

6. Select **View → Pin Legend Window** to display the pin legend.
   Close the **Pin Planner**.

7. **Export Pin Assignments**
   With the **Assignment Editor** selected, select **Assignments → Export Assignments** to create a .qsf file containing your pin assignments. This .qsf file can be used for other projects by selecting **Assignments → Import Assignments** when assigning pins.

Task 4: Using the TimeQuest Timing Analyzer

1. Select **Assignments → Settings**
   Click the **Timing Analysis Settings** and verify that **Use TimeQuest Timing Analyzer during compilation** is selected for **Timing analysis processing**.
   Click **OK**.



Timing analysis processing

● Use TimeQuest Timing Analyzer during compilation

○ Use Classic Timing Analyzer during compilation

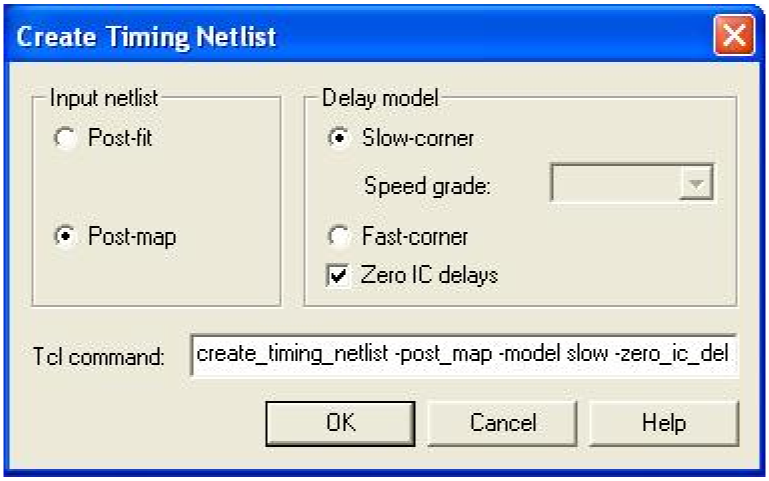
2. **Synthesize the Design**

   Select the **Start Analysis & Synthesis** button from the Toolbar
   Click **OK** to continue when the analysis and synthesis is complete.

3. **Open the TimeQuest Timing Analzyer**

   Select **Tools → TimeQuest Timing Analyzer**, or use the Toolbar button
   Click **No** when the dialog box pops up asking if you would like to generate an SDC file from the Quartus Settings File.

4. **Create a Timing Netlist**
   In the TimeQuest Timing Analyzer window, select **Netlist → Create Timing Netlist.**
   Set the **Input netlist type** to **Post-map** and click **OK.**



Create Timing Netlist

Input netlist
○ Post-fit
● Post-map

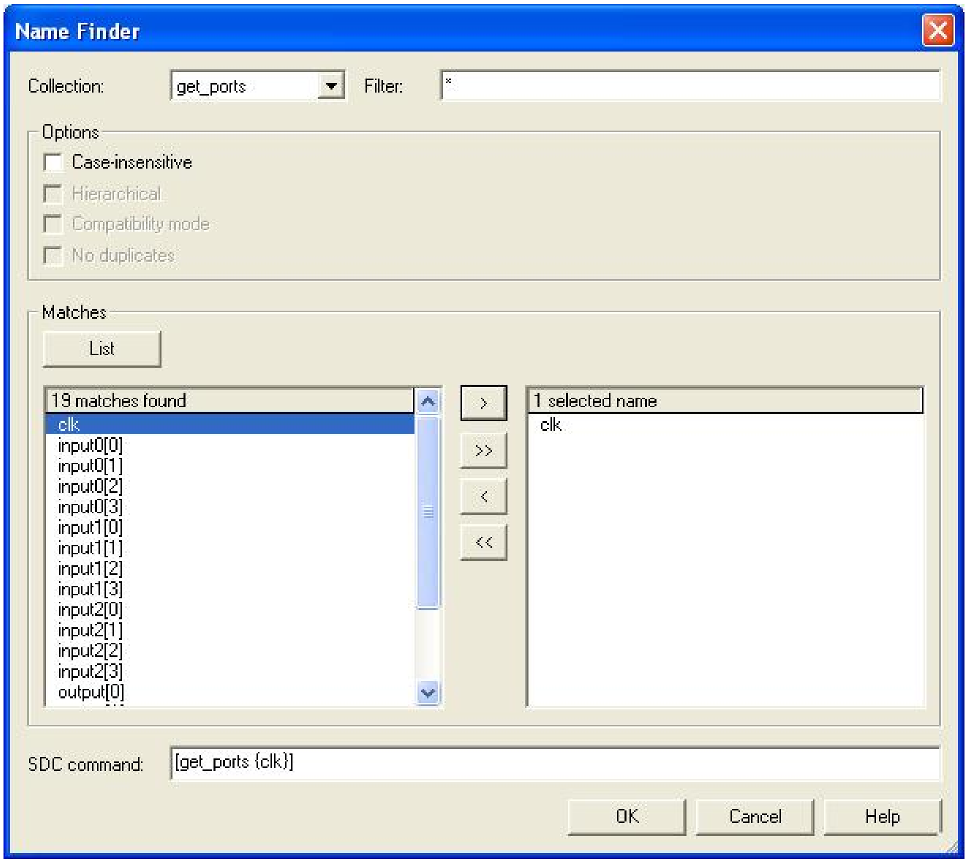
Delay model
● Slow-corner
Speed grade:
○ Fast-corner
☑ Zero IC delays

Tcl command: create_timing_netlist -post_map -model slow -zero_ic_del

5. Select **Constraints → Create Clock...** from the Menu Bar.
   Type **clk** for the **Clock name**, and a **Period** of **20 ns**.
   Double Click **Read SDC File** in the **Tasks** window to read in an SDC file.

   Click the ellipses ... to select the **Targets**, which opens up the **Name Finder** window.
   Click **List** in the **Matches** field and select **clk.**

   Click the single right arrow > to add it to the list of selected names and then click **OK.**
   Click **Run** in the **Create Clock** dialog box to create the clock.



Name Finder

Collection: get_ports
Filter: *

Options
☐ Case-insensitive
☐ Hierarchical
☐ Compatibility mode
☐ No duplicates

Matches

19 matches found
clk
input0[0]
input0[1]
input0[2]
input0[3]
input1[0]
input1[1]
input1[2]
input1[3]
input2[0]
input2[1]
input2[2]
input2[3]
output[0]

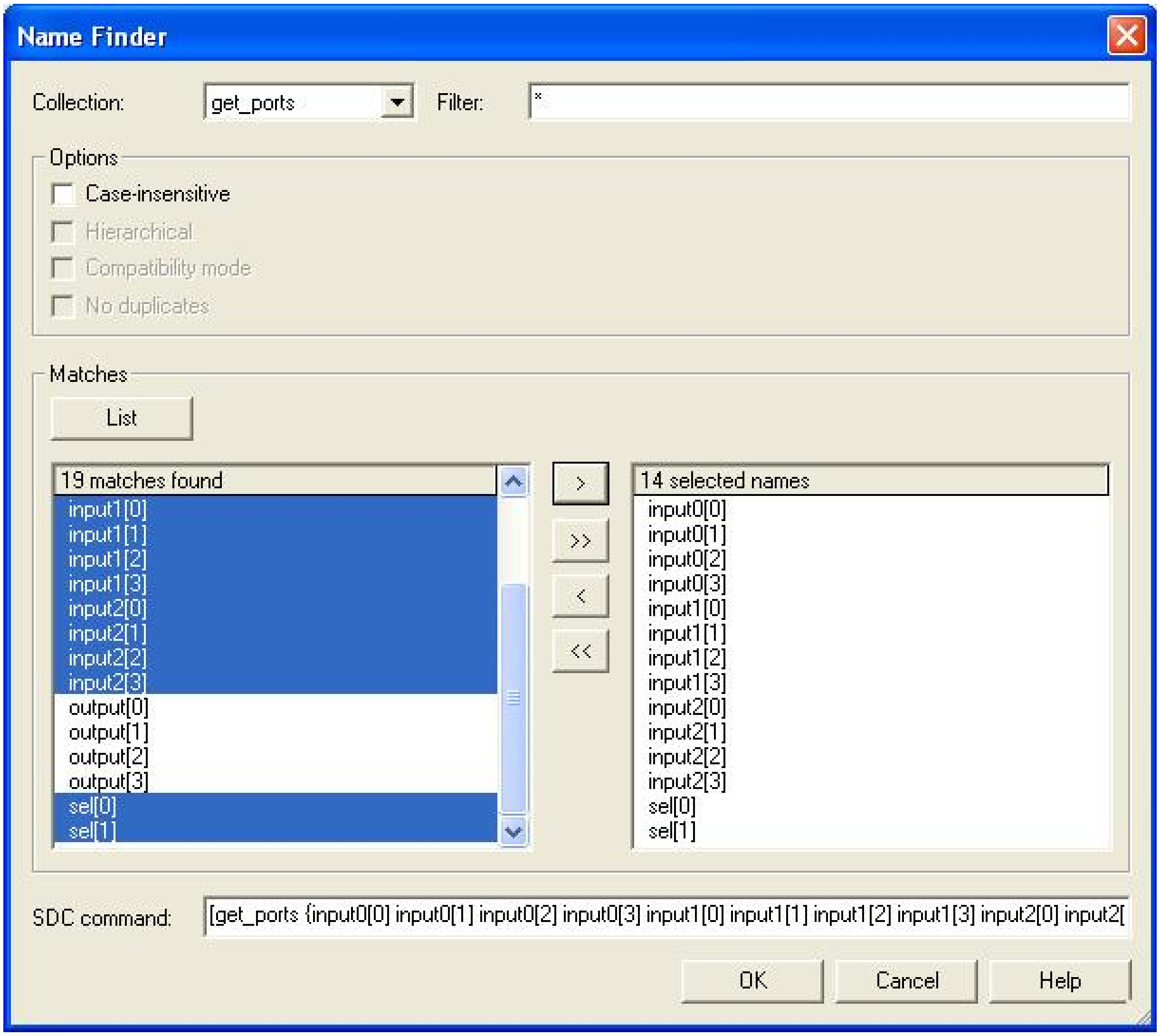
1 selected name
clk

SDC command: [get_ports {clk}]

6. Select **Constraints → Set Input Delay…** from the Menu Bar.
   For **Clock name:** select **clk**.
   Make a **Delay value** of **20 ns**.

   Click the ellipses [...] to select the T**argets**.
   Click **List** in the **Matches** field and select all of the **input** and **sel** names.

   Click the single right arrow [>] to add them to the list of selected names and then click **OK**.
   Click **Run** in the Set Input Delay window to create the input timing constraints.



Name Finder

Collection: get_ports   Filter: *

Options
- □ Case-insensitive
- □ Hierarchical
- □ Compatibility mode
- □ No duplicates

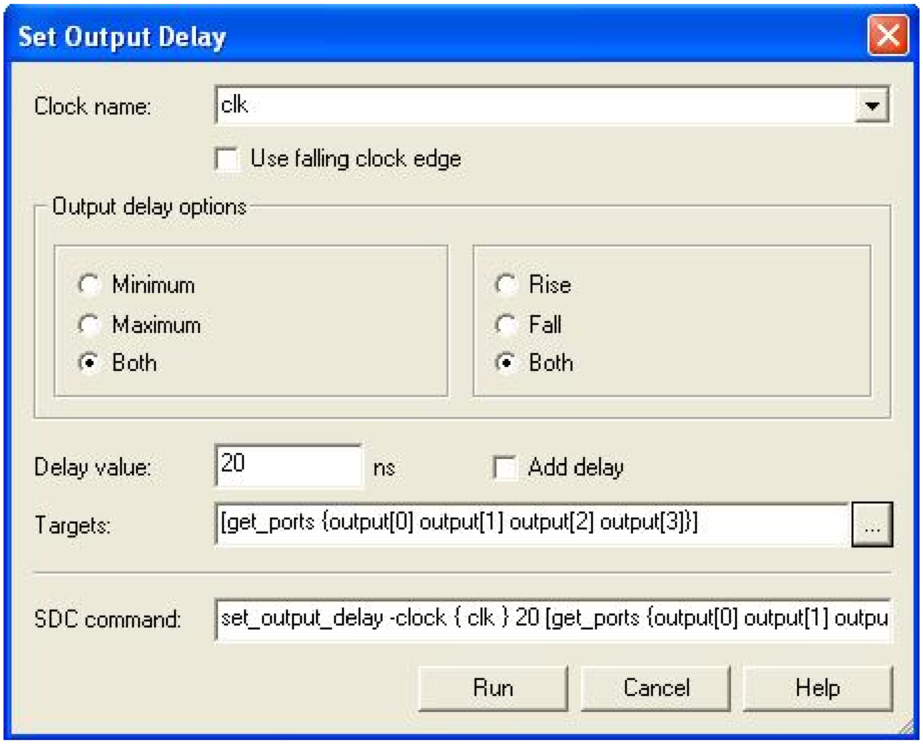
Matches

19 matches found	14 selected names
input1[0]	input0[0]
input1[1]	input0[1]
input1[2]	input0[2]
input1[3]	input0[3]
input2[0]	input1[0]
input2[1]	input1[1]
input2[2]	input1[2]
input2[3]	input1[3]
output[0]	input2[0]
output[1]	input2[1]
output[2]	input2[2]
output[3]	input2[3]
sel[0]	sel[0]
sel[1]	sel[1]

SDC command: [get_ports {input0[0] input0[1] input0[2] input0[3] input1[0] input1[1] input1[2] input1[3] input2[0] input2[

7. Select **Constraints** → **Set Output Delay…** from the Menu Bar.
   For **Clock name:** select **clk**.
   Make a **Delay value** of **20 ns**.

8. Click the ellipses ... to select the **Targets**.
   Click **List** in the **Matches** field and select all of the **output** names.
   Click the single right arrow > to add it to the list of selected names and then click **OK**.
   Click **Run** in the **Set Input Delay** window to create the output timing constraints.



Clock name: clk

☐ Use falling clock edge

Output delay options

○ Minimum
○ Maximum
● Both

○ Rise
○ Fall
● Both

Delay value: 20 ns ☐ Add delay

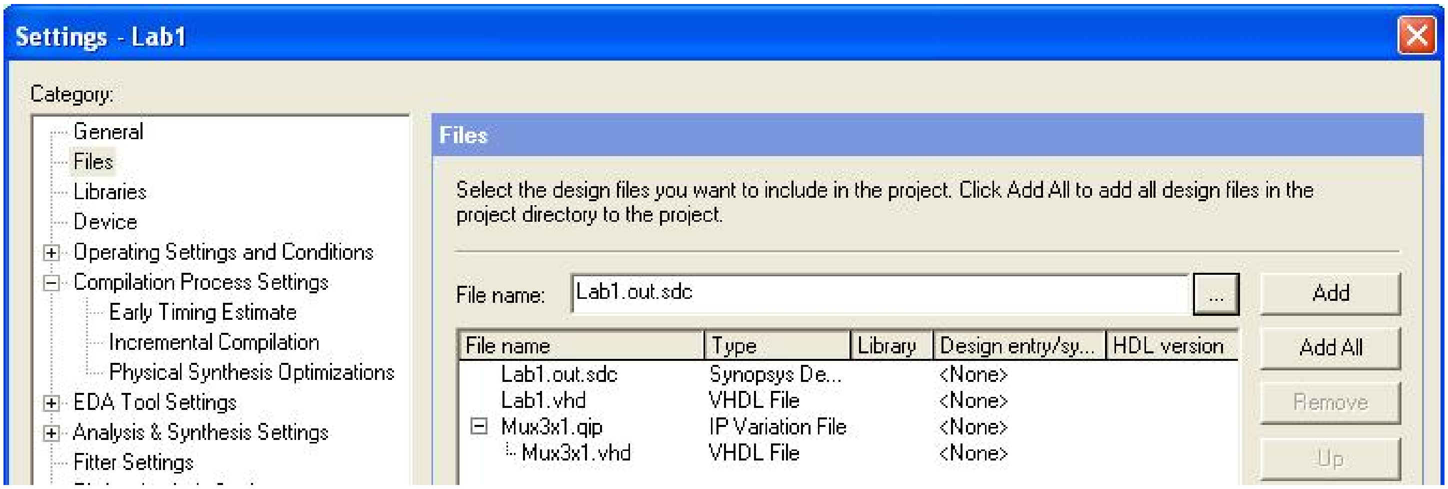
Targets: [get_ports {output[0] output[1] output[2] output[3]}]

SDC command: set_output_delay -clock { clk } 20 [get_ports {output[0] output[1] outpu

9. Select **Constraints** → **Write SDC File** from the Menu Bar to save the SDC file.
   Click **Save** in the **Save As** dialog box.

10. Close the TimeQuest Timing Analyzer.

11. **Add the SDC File to Your Project**
    In the Quartus II window, select **Project → Add/Remove Files in Project…** from the Menu Bar.
    Click the ellipses next to **File name:** in the **Settings** window that popped up.
    Select **Script Files** in the drop-down box for **Files of type** in the **Select File** window.
    Select the **Lab1.out.sdc** file and click **Open**.
    Click **Add** and then **OK** in the **Settings** window.



Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

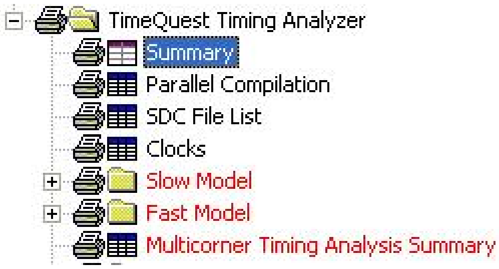
File name: Lab1.out.sdc

File name	Type	Library	Design entry/sy...	HDL version
Lab1.out.sdc	Synopsys De...		<None>	
Lab1.vhd	VHDL File		<None>	
Mux3x1.qip	IP Variation File		<None>	
Mux3x1.vhd	VHDL File		<None>	

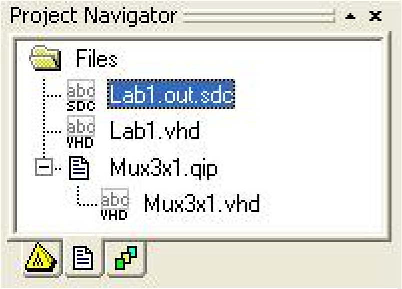
12. Compile your design by clicking the Start Compilation button on the toolbar ▶

13. Click the **Critical Warning** tab in the **Messages** window to see that the timing requirements have not been met. This is because of the long input and output delays specified.

14. Expand the **TimeQuest Timing Analyzer** category in the **Compilation** Report and note that the **Slow Model, Fast Model, and Multicorner Timing Analysis Summary** reports are in red, indicating that they have failed the timing analysis. Expanding each of the red submenus will show the failures in more detail.



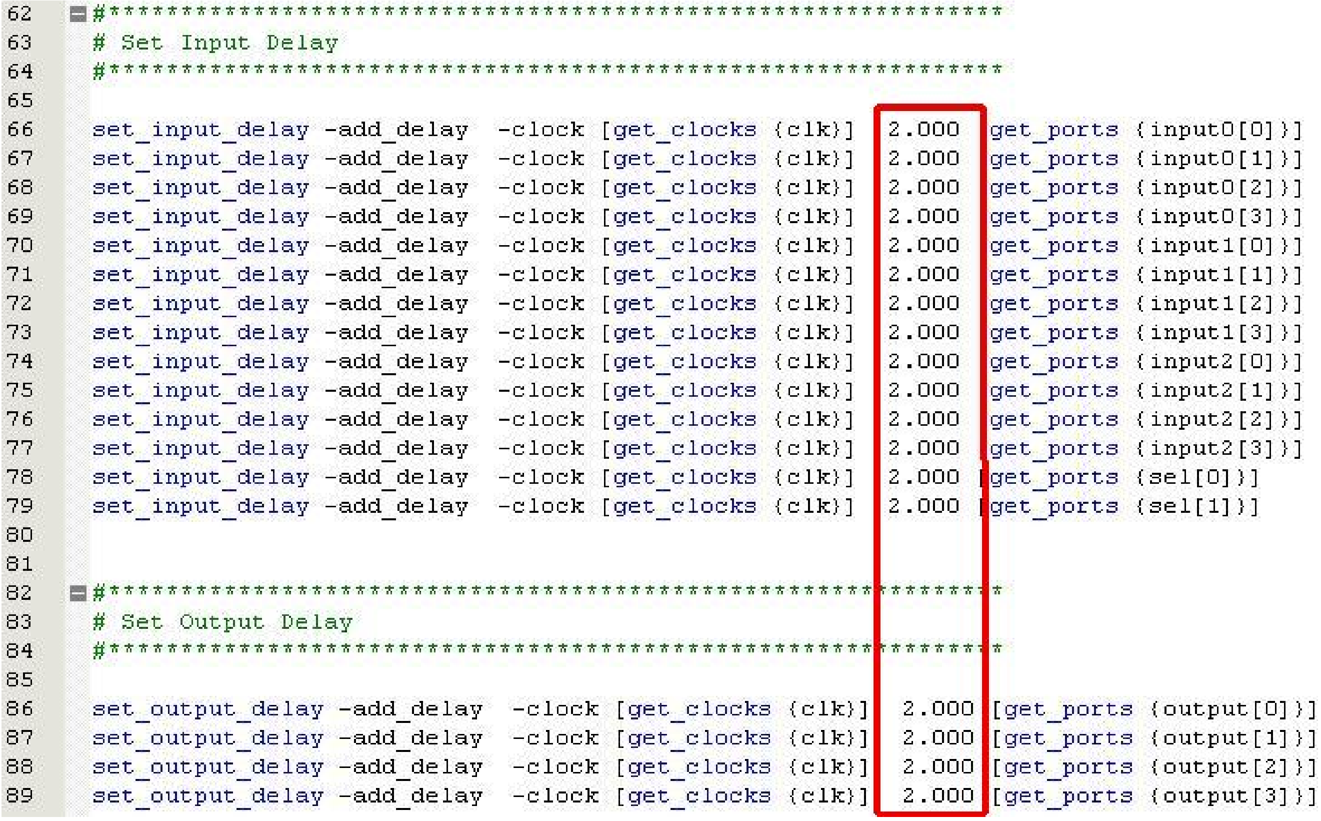
15. In the **Files** section of the **Project Navigator**, double-click on **Lab1.out.sdc** to directly edit the SDC file.



16. Scroll down in the **Lab1.out.sdc** file to find the **Set Input Delay** and **Set Output Delay** sections. These should be starting on lines 63 and 83 for this example. For each input line that reads similar to:

```
set_input_delay -add_delay  -clock [get_clocks {clk}]  20.000 [get_ports {input0[0]}]
```

Change the **20.000** ns delay to **2.000** ns. Repeat this process for all the input and delays.



```
62  #**************************************************************
63  # Set Input Delay
64  #**************************************************************
65
66  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input0[0]}]
67  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input0[1]}]
68  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input0[2]}]
69  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input0[3]}]
70  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input1[0]}]
71  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input1[1]}]
72  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input1[2]}]
73  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input1[3]}]
74  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input2[0]}]
75  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input2[1]}]
76  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input2[2]}]
77  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {input2[3]}]
78  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {sel[0]}]
79  set_input_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {sel[1]}]
80
81
82  #**************************************************************
83  # Set Output Delay
84  #**************************************************************
85
86  set_output_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {output[0]}]
87  set_output_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {output[1]}]
88  set_output_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {output[2]}]
89  set_output_delay -add_delay  -clock [get_clocks {clk}]  2.000 [get_ports {output[3]}]
```

Save the Lab1.out.sdc file by clicking on the **Save** button on the Menu Bar

17. Compile the design again by clicking the Start Compilation button on the toolbar and check the TimeQuest Timing Analyzer reports again to see that the design has now passed the timing analysis. Ignore any warnings at this time.

Task 5: Program the DE2 Development Board

1. On the Altera DE2 Development board, plug the USB Cable into the **USB Blaster Port**. Plug the other end of the USB Cable into the computer.
   Plug the 9v power supply into the **9V DC Power Supply Connector** on the DE2 board, and plug the other end into a 120v wall socket.
   Press the **Power ON/OFF Switch** to power on the DE2 board.
   Make sure that the **RUN/PROG Switch for JTAG/AS Modes** is in the **RUN** position.

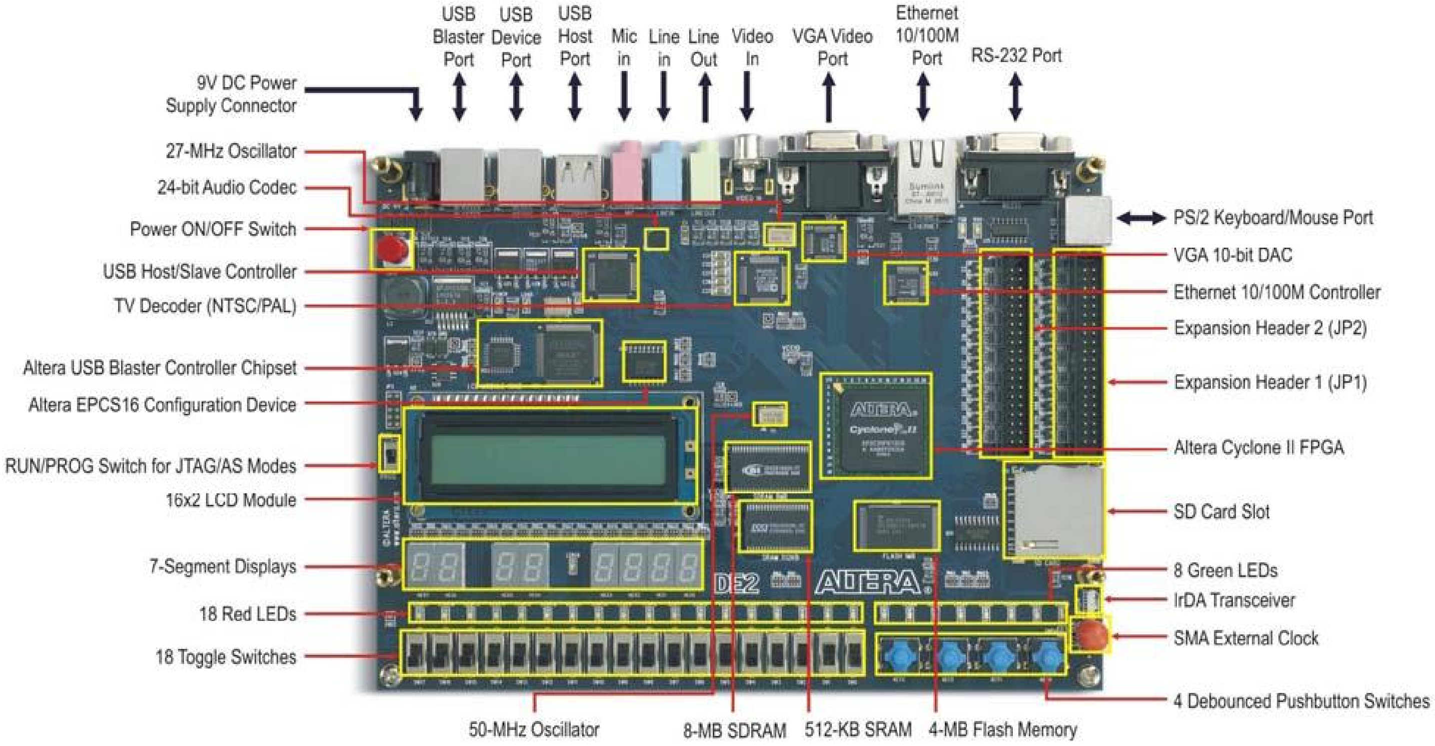
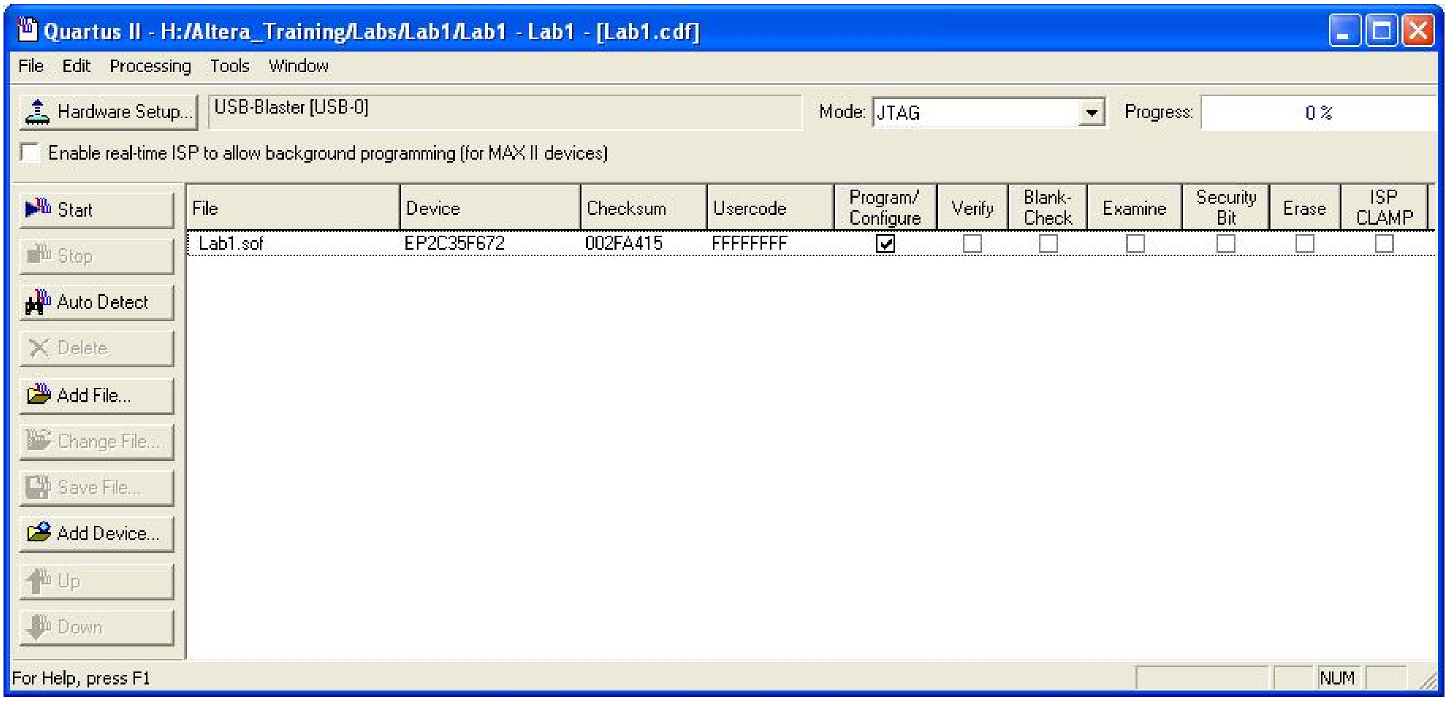


Image Source: DE2 Development and Education Board User Manual version 1.42 , Altera Corperation, 2008

2. In the Quartus II window click the **Programmer** button on the Toolbar to open the Programmer window
   The **Hardware Setup…** must be **USB-Blaster [USB-0]**. If not, click the **Hardware Setup…** button and select **USB-Blaster [USB-0]** from the drop-down menu for **Currently selected hardware**.
   Mode should be set to **JTAG**.
   Make sure that the **File** is **Lab1.sof**, **Device** is **EP2C35F672**, and the **Program/Configure** box is checked.



Quartus II - H:/Altera_Training/Labs/Lab1/Lab1 - Lab1 - [Lab1.cdf]

Hardware Setup... USB-Blaster [USB-0]   Mode: JTAG   Progress: 0 %

□ Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
Lab1.sof	EP2C35F672	002FA415	FFFFFFFF	☑	□	□	□	□	□	□

For Help, press F1   NUM

   Then click the **Start** button to program the DE2 board.
   When the progress bar reaches 100%, programming is complete.

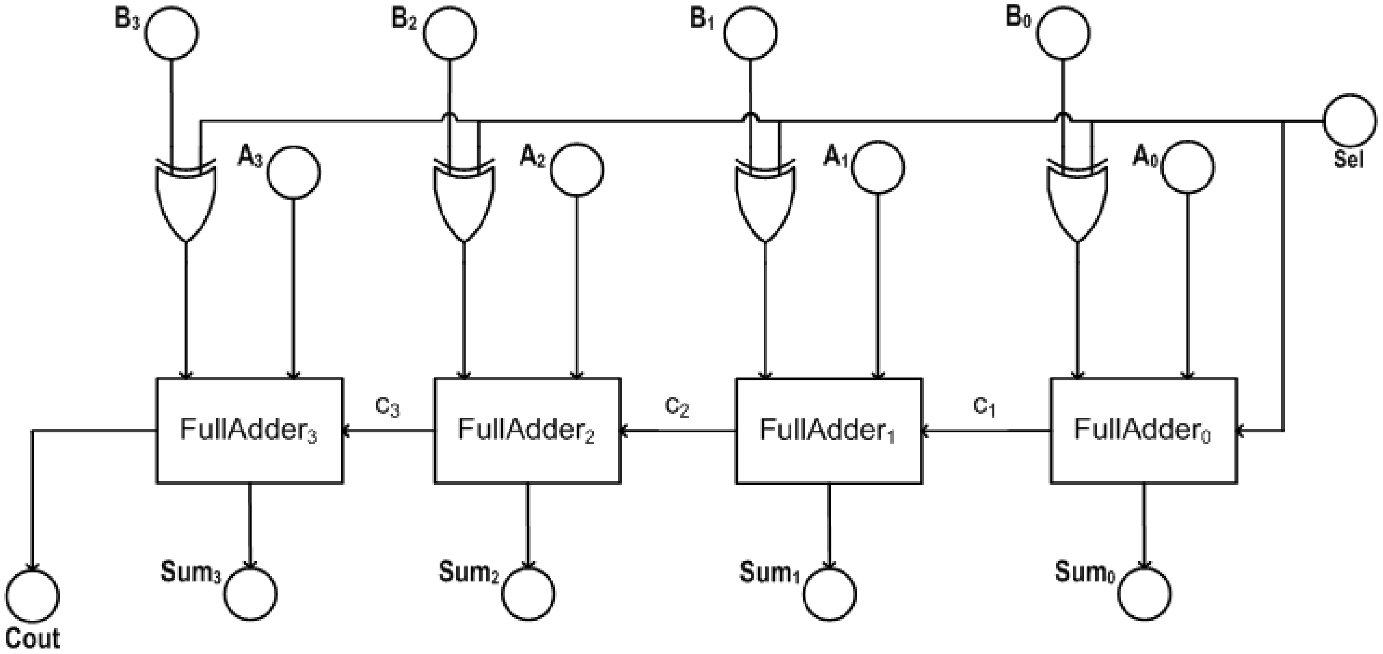
3. You can now test the program on the DE2 board by using the toggle switches located along the bottom of the board.
   SW0 and SW1 are the selector inputs to the multiplexer.
   SW4 through SW7 are the four bits of data for input0.
   SW9 through SW11 are the four bits of data for input1.
   SW13 through SW17 are the four bits of data for input2.
   The output of the multiplexer is displayed on the first four green LEDs located above the blue push buttons on the DE2 board. Their pattern will correspond to the four bits of the selected data stream.

Lab 2

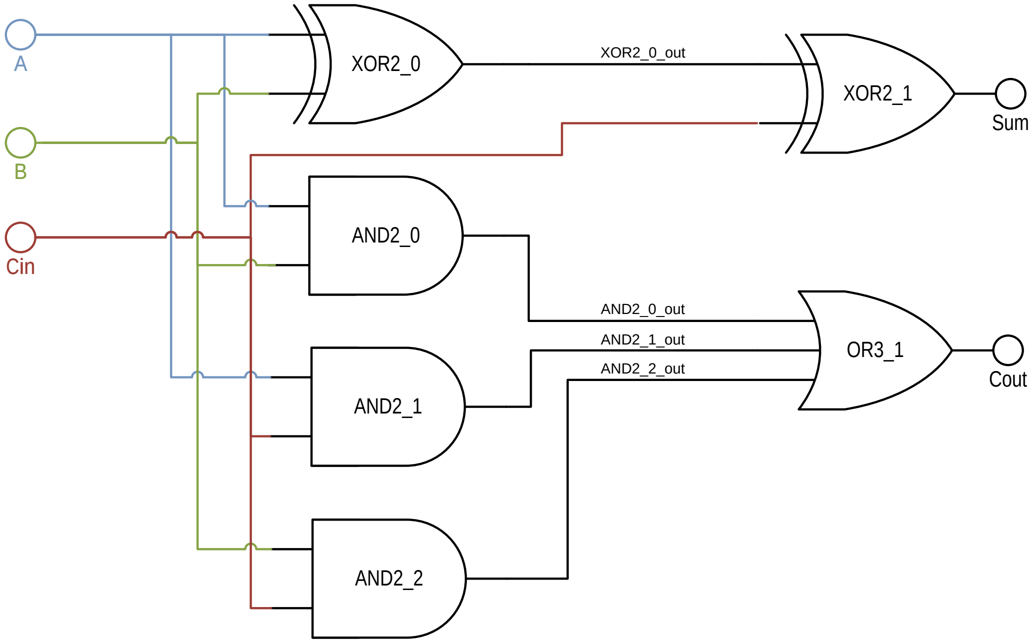
Introduction to VHDL

Lab 2 – Introduction to VHDL

In this lab you will design, test, and simulate a basic logic circuit using the Quartus II development software. The circuit you will create is a ripple-carry four-bit adder/subtractor using both behavioral and structural VHDL coding. Below is a schematic diagram of the complete circuit. The component make-up includes four full adders and four XOR logic gates. The inputs are two four-bit numbers; A and B, and an add/subtract selection input; Sel. The outputs are four-bit sum; Sum, and a carry-out output; Cout.



Below is a detailed diagram of the full adder circuit used. It consists of three two-input AND gates, two XOR gates, and one three-input OR gate. The individual gates will be coded separately using behavioral style VHDL, and then stitched together using structural style VHDL to create the full adder.



Task 1: Create a New Project

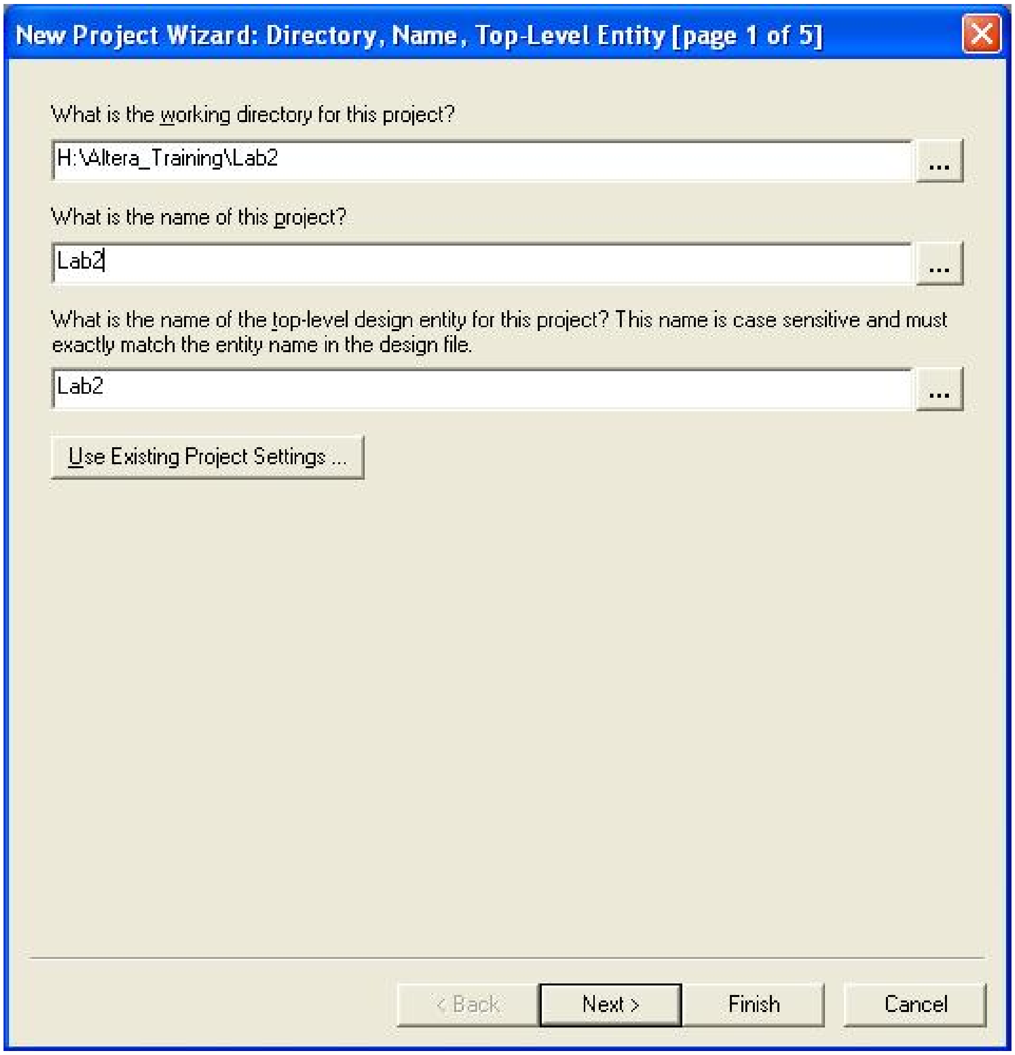
1. **Start the Quartus II software.**
   From the Windows Start Menu, select:
   All Programs → Other Apps → Altera → Quartus II 9.1 → Quartus II 9.1 (32-Bit)

2. **Start the New Project Wizard.**
   If the opening splash screen is displayed, select: **Create a New Project (New Project Wizard)**, otherwise from the Quartus II Menu Bar select: **File → New Project Wizard**.

3. **Select the Working Directory and Project Name.**

Working Directory	H:\Altera_Training\Lab2
Project Name	Lab2
Top-Level Design Entity	Lab2

   Click **Next** to advance to page 2 of the New Project Wizard.
   Note: A window may pop up stating that the chosen working directory does not exist. Click ***Yes*** *to create it.*



New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

H:\Altera_Training\Lab2

What is the name of this project?

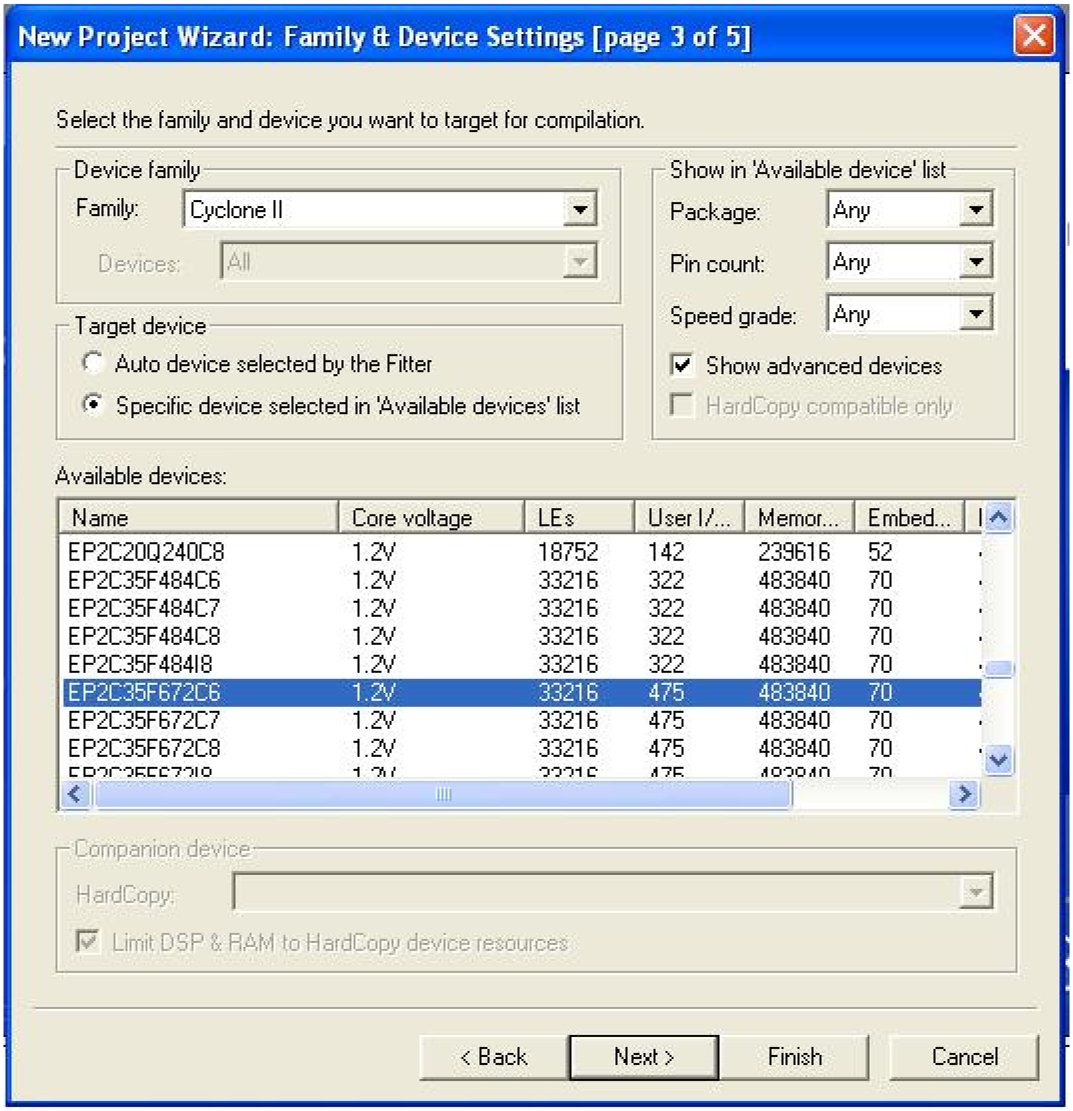
Lab2

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

Lab2

4. Click **Next** again as we will not be adding any preexisting design files at this time.

5. **Select the family and Device Settings.**
   From the pull-down menu labeled **Family**, select **Cyclone II**.
   In the list of available devices, select **EPC235F672C6.**
   Click **Next.**



New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family
Family: Cyclone II
Devices: All

Show in 'Available device' list
Package: Any
Pin count: Any
Speed grade: Any
☑ Show advanced devices
☐ HardCopy compatible only

Target device
○ Auto device selected by the Fitter
● Specific device selected in 'Available devices' list

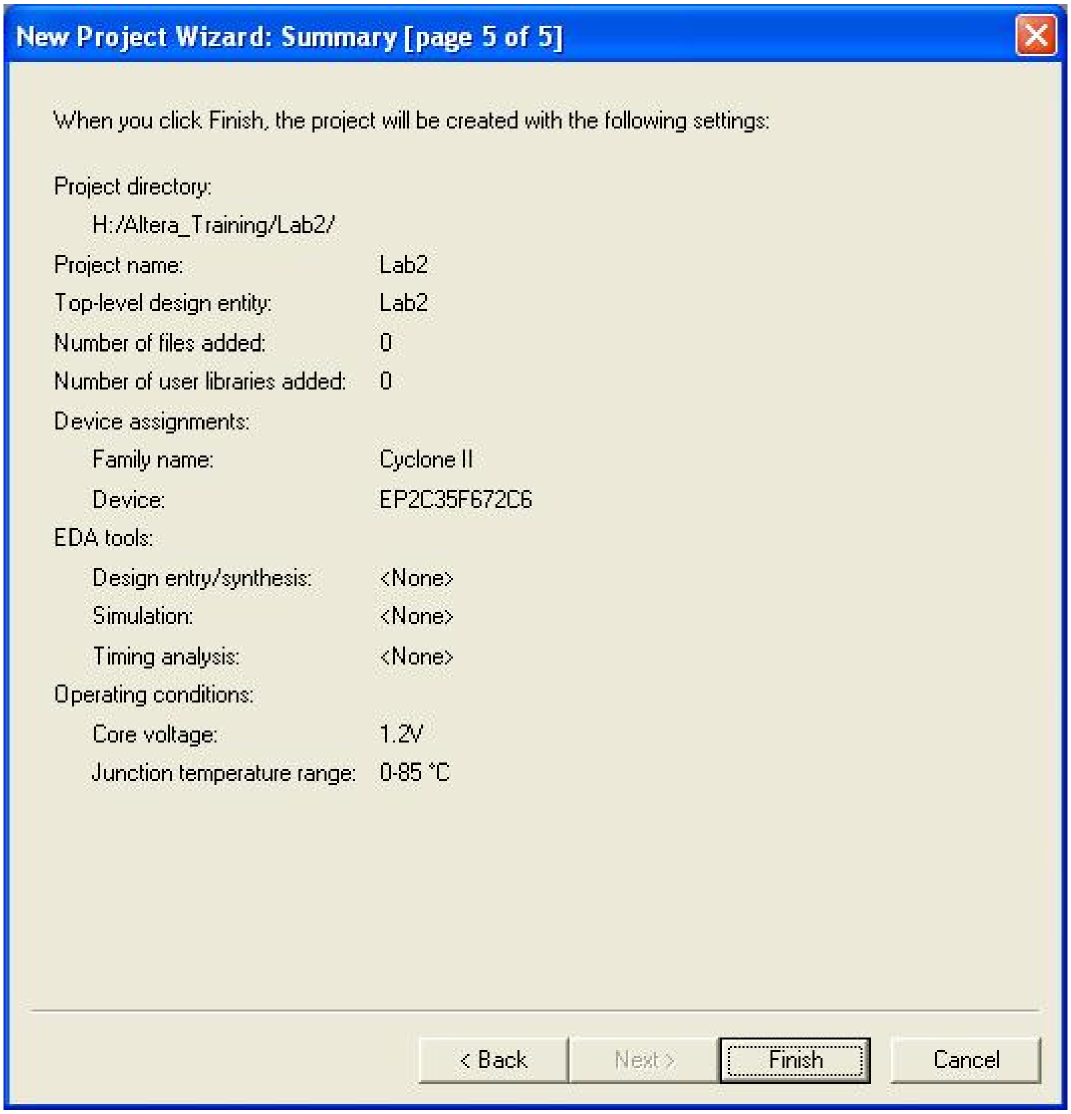
Available devices:

Name	Core voltage	LEs	User I/...	Memor...	Embed...	I
EP2C20Q240C8	1.2V	18752	142	239616	52	
EP2C35F484C6	1.2V	33216	322	483840	70	
EP2C35F484C7	1.2V	33216	322	483840	70	
EP2C35F484C8	1.2V	33216	322	483840	70	
EP2C35F484I8	1.2V	33216	322	483840	70	
EP2C35F672C6	1.2V	33216	475	483840	70	
EP2C35F672C7	1.2V	33216	475	483840	70	
EP2C35F672C8	1.2V	33216	475	483840	70	
EP2C35F672I8	1.2V	33216	475	483840	70	

Companion device
HardCopy:
☑ Limit DSP & RAM to HardCopy device resources

6. Click **Next** again as we will not be using any third party EDA tools.

7. Click **Finish** to complete the New Project Wizard.



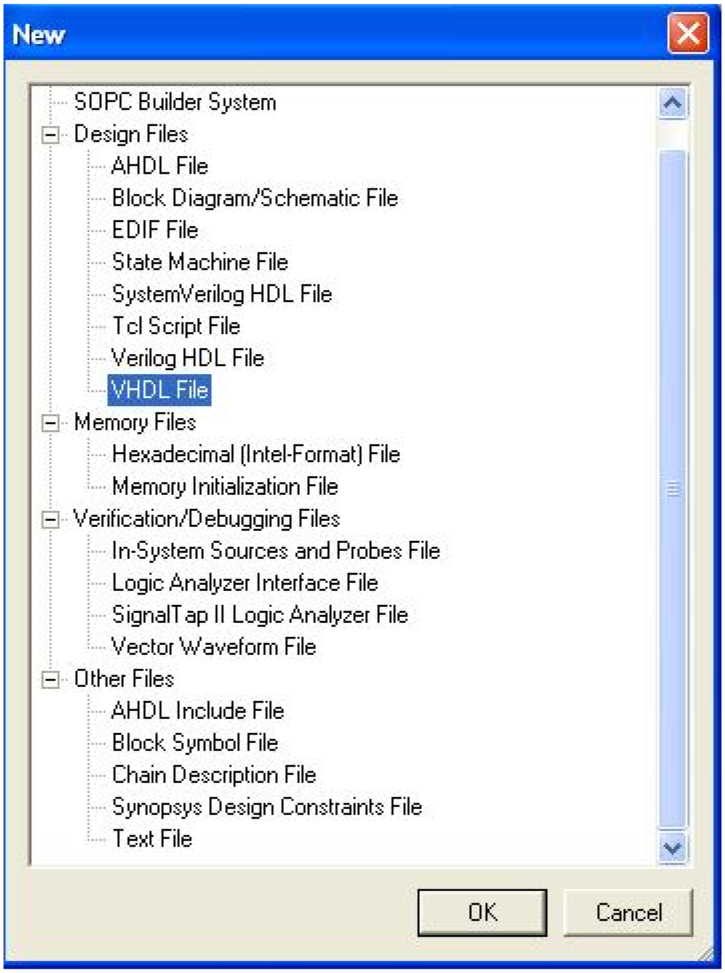
New Project Wizard: Summary [page 5 of 5]

When you click Finish, the project will be created with the following settings:

Project directory:	
H:/Altera_Training/Lab2/	
Project name:	Lab2
Top-level design entity:	Lab2
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone II
Device:	EP2C35F672C6
EDA tools:	
Design entry/synthesis:	<None>
Simulation:	<None>
Timing analysis:	<None>
Operating conditions:	
Core voltage:	1.2V
Junction temperature range:	0-85 °C

Task 2: Create, Add, and Compile Design Files

1. **Create a new Design File.**
   Select: **File → New** from the Menu Bar.
   Select: **VHDL File** from the **Design Files** list and click **OK**.



2. Copy and paste the following code into your new VHDL file, then save it by selecting **File** → **Save**. Name the file **And2** and click **Save** in the **Save As** dialog box.

Lab 2 – Introduction to VHDL – Two-Input AND Gate Behavioral VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Gate_And2 IS
   PORT (x: IN std_logic;
          y: IN std_logic;
         F: OUT std_logic);
END Gate_And2;

ARCHITECTURE Gate_And2_beh OF Gate_And2 IS
BEGIN
   PROCESS(x, y)
   BEGIN
      F <= x AND y;
   END PROCESS;
END Gate_And2_beh;
```

This is a basic two-input AND Gate, written using Behavioral style VHDL. The AND gate has two inputs, **x** and **y**, which are ANDed together and the result of the logic operation is returned on **F**. The **Process** sensitivity list includes **x** and **y**, so whenever either of the two change in value the process will execute and generate a new result.

3. Create another new VHDL file following the directions in step 1 of task 2.

4. Copy and paste the following code into your new VHDL file, then save it by selecting **File** → **Save**. Name the file **Xor2** and click **Save** in the **Save As** dialog box.

Lab 2 – Introduction to VHDL – Two-Input XOR Gate Behavioral VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Gate_XOR2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END Gate_XOR2;

ARCHITECTURE Gate_XOR2_beh OF Gate_XOR2 IS
BEGIN
   PROCESS(x, y)
   BEGIN
      F <= x XOR y;
   END PROCESS;
END Gate_XOR2_beh;
```

This is a basic XOR Gate, written using Behavioral style VHDL. Similar to the AND gate we created, the XOR gate has two inputs, **x** and **y**, which are XORed together and the result of the logic operation is returned on **F**. The **Process** sensitivity list includes **x** and **y**, so whenever either of the two change in value the process will execute and generate a new result.

5. Create another new VHDL file following the directions in step 1 of task 2.

6. Copy and paste the following code into your new VHDL file, then save it by selecting **File** → **Save**. Name the file **Or3** and click **Save** in the **Save As** dialog box. This is a three-input OR Gate, written using Behavioral style VHDL.

Lab 2 – Introduction to VHDL – Three-Input OR Gate Behavioral VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Gate_OR3 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         z: IN std_logic;
         F: OUT std_logic);
END Gate_OR3;

ARCHITECTURE Gate_OR3_beh OF Gate_OR3 IS
BEGIN
   PROCESS(x, y, z)
   BEGIN
      F <= x OR y OR z;
   END PROCESS;
END Gate_OR3_beh;
```

This is a three-input OR Gate, written using Behavioral style VHDL. Similar to the AND gate and XOR gatewe created, the OR gate has three inputs, **x, y,** and **z**, which are ORed together and the result of the logic operation is returned on **F**. The **Process** sensitivity list includes **x**, **y**, and **z**, so whenever one changes in value the process will execute and generate a new result.

7. Create another new VHDL file following the directions in step 1 of task 2.

8. Copy and paste the following code into your new VHDL file, then save it by selecting **File → Save**. Name the file **FullAdder** and click **Save** in the **Save As** dialog box.

Lab 2 – Introduction to VHDL – Full Adder Structural VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY FullAdder IS
   PORT (A: IN std_logic;
         B: IN std_logic;
         Cin: IN std_logic;
         Sum: OUT std_logic;
         Cout: OUT std_logic);
END FullAdder;

ARCHITECTURE FullAdder_struct OF FullAdder IS

      COMPONENT Gate_And2 IS
            PORT (x, y: IN std_logic;
                  f: OUT std_logic);
      END COMPONENT;

      COMPONENT Gate_XOR2 IS
            PORT (x, y: IN std_logic;
                  f: OUT std_logic);
      END COMPONENT;

      COMPONENT Gate_OR3 IS
            PORT (x, y, z: IN std_logic;
                  f: OUT std_logic);
      END COMPONENT;

      SIGNAL XOR2_0_out, AND2_0_out, AND2_1_out, AND2_2_out: std_logic;

BEGIN

      XOR2_0: Gate_XOR2 PORT MAP(A, B, XOR2_0_out);
      XOR2_1: Gate_XOR2 PORT MAP(XOR2_0_out, Cin, Sum);
      AND2_0: Gate_AND2 PORT MAP(A, B, AND2_0_out);
      AND2_1: Gate_AND2 PORT MAP(A, Cin, AND2_1_out);
      AND2_2: Gate_AND2 PORT MAP(B, Cin, AND2_2_out);
      OR3_1:  Gate_OR3 PORT MAP(AND2_0_out, AND2_1_out, AND2_2_out, Cout);

END FullAdder_struct;
```

This is an example of structural-style VHDL which utilizes the multiple components we previously created and stitches them together to produce the desired result. In this case there are two instances of the XOR gate used, three AND gates used, and one OR gate to create the full adder. The argument list for each component instance is fashioned in a way to produce the desired connections based on the schematic diagram of the adder/subtractor circuit. A **SIGNAL** declaration is used to define the connections that are internal to the circuit. These signals carry the signal from one component to another.

9. Create another new VHDL file following the directions in step 1 of task 2.

10. Copy and paste the following code into your new VHDL file, then save it by selecting **File** → **Save**. Name the file **Lab2** and click **Save** in the **Save As** dialog box. This is the Full Adder circuit, written using Structural style VHDL.

Lab 2 – Introduct on to VHDL – Adder/Subtractor Structural VHDL Code

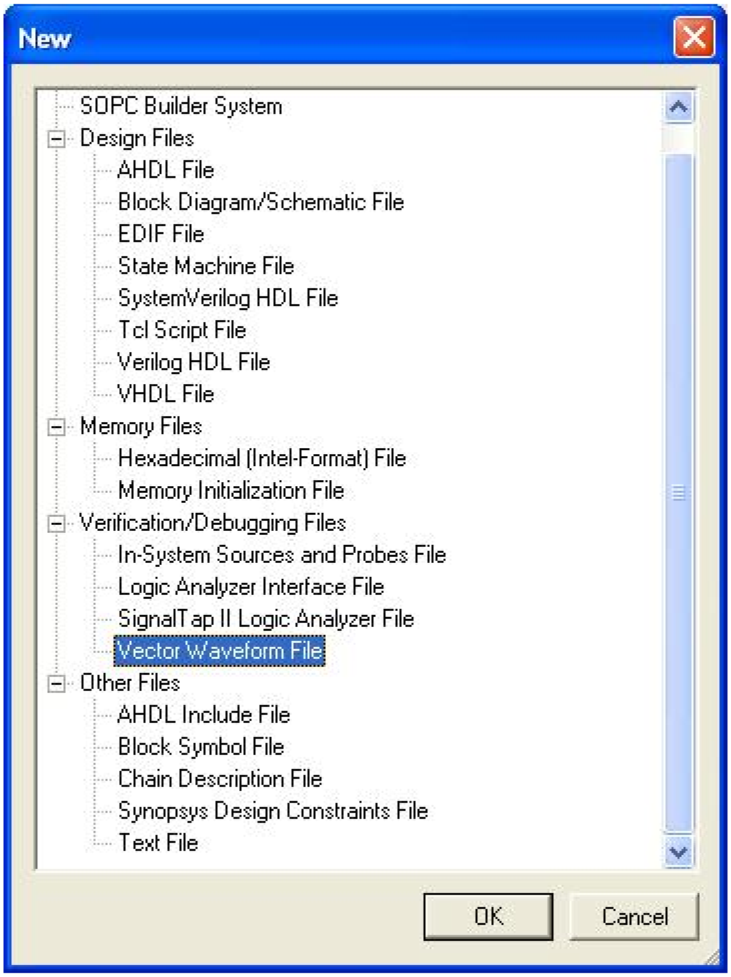
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Lab2 IS
   PORT (A: IN std_logic_vector(3 DOWNTO 0);
         B: IN std_logic_vector(3 DOWNTO 0);
         Sel: IN std_logic;
         Sum: OUT std_logic_vector(3 DOWNTO 0);
         Cout: OUT std_logic);
END Lab2;

ARCHITECTURE AddSub_struct OF Lab2 IS

      COMPONENT Gate_XOR2 IS
            PORT (x, y: IN std_logic;
                  f: OUT std_logic);
      END COMPONENT;

      COMPONENT FullAdder IS
            PORT (A: IN std_logic;
             B: IN std_logic;
             Cin: IN std_logic;
             Sum: OUT std_logic;
             Cout: OUT std_logic);
      END COMPONENT;

      SIGNAL XOR2_0_out, XOR2_1_out, XOR2_2_out, XOR2_3_out: std_logic;
      SIGNAL c1, c2, c3: std_logic;

BEGIN

      XOR2_0: Gate_XOR2 PORT MAP(B(0), Sel, XOR2_0_out);
      XOR2_1: Gate_XOR2 PORT MAP(B(1), Sel, XOR2_1_out);
      XOR2_2: Gate_XOR2 PORT MAP(B(2), Sel, XOR2_2_out);
      XOR2_3: Gate_XOR2 PORT MAP(B(3), Sel, XOR2_3_out);
      FullAdder_0: FullAdder PORT MAP(A(0), XOR2_0_out, Sel, Sum(0), c1);
      FullAdder_1: FullAdder PORT MAP(A(1), XOR2_1_out, c1, Sum(1), c2);
      FullAdder_2: FullAdder PORT MAP(A(2), XOR2_2_out, c2, Sum(2), c3);
      FullAdder_3: FullAdder PORT MAP(A(3), XOR2_3_out, c3, Sum(3), Cout);

END AddSub_struct;
```

11. Compile the design by clicking the **Start Compilation** button ▶ on the Toolbar.

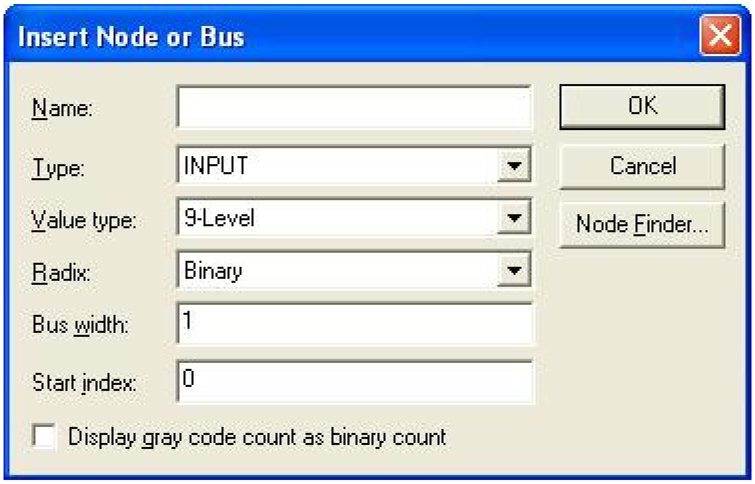
Task 3: Simulate Design using Quartus II

1. **Create a Vector Waveform File (vwf)**

   Click the **New File** icon on the Menu Bar
   Select **Vector Waveform File** under the **Verification/Debugging Files** heading.



2. **Add Signals to the Waveform**
   Select **Edit → Insert → Insert Node or Bus…** to open the **Insert Node or Bus** window.



Name:

Type: INPUT

Value type: 9-Level

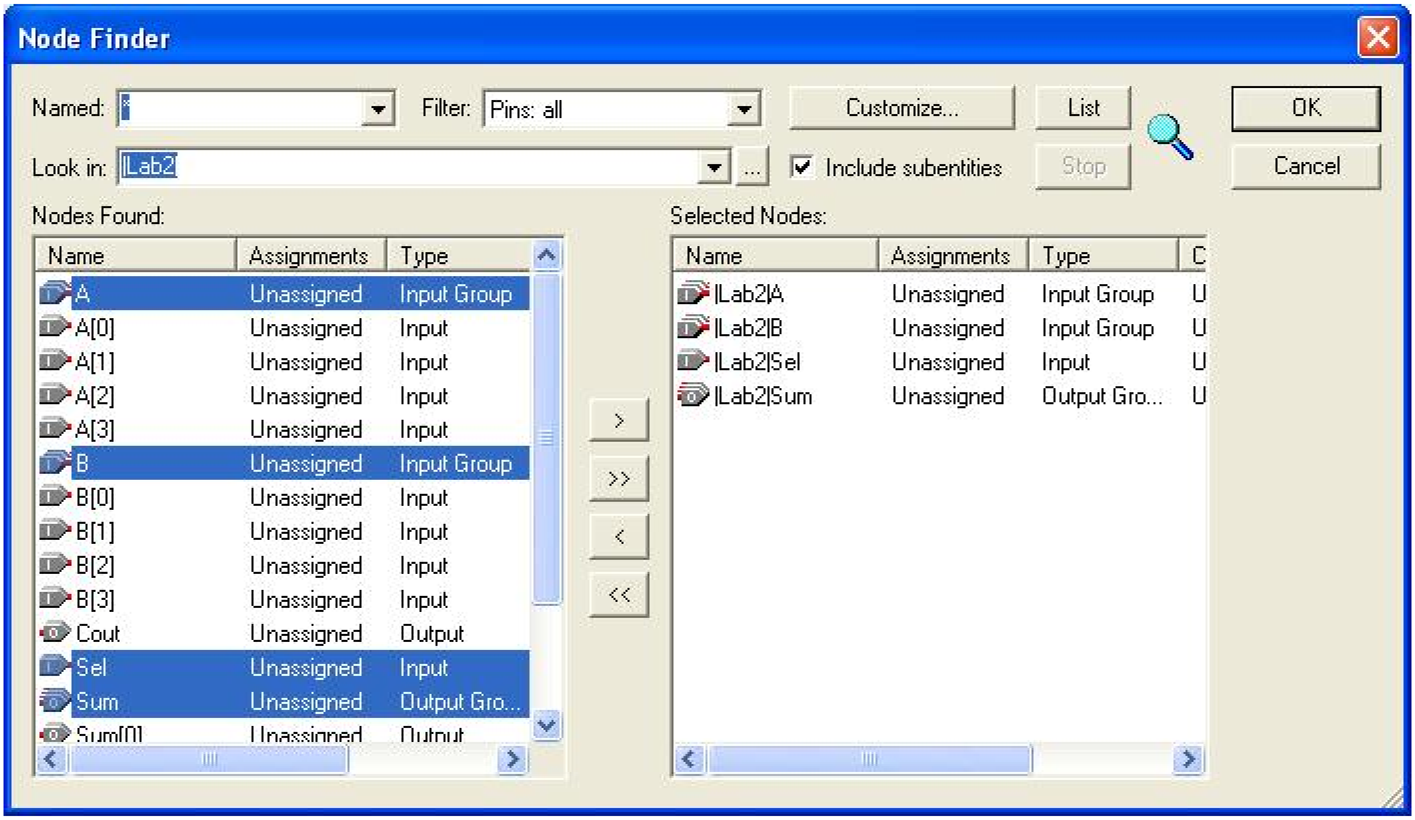
Radix: Binary

Bus width: 1

Start index: 0

☐ Display gray code count as binary count

3. Click the **Node Finder…** button in the **Insert Node or Bus** window to open the **Node Finder** window.
From the **Filter** drop-down menu, select **Pins: all** and click the **List** button. This will display all the inputs and outputs to the circuit.
Highlight all the the **Input Groups A,** and **B**, the **Input Sel**, and **Output Group** Sum in the **Nodes Found** window and click the right arrow > to select them.
Then click **OK** to close the **Node Finder** window.



Named: *

Filter: Pins: all

Look in: |Lab2|

☑ Include subentities

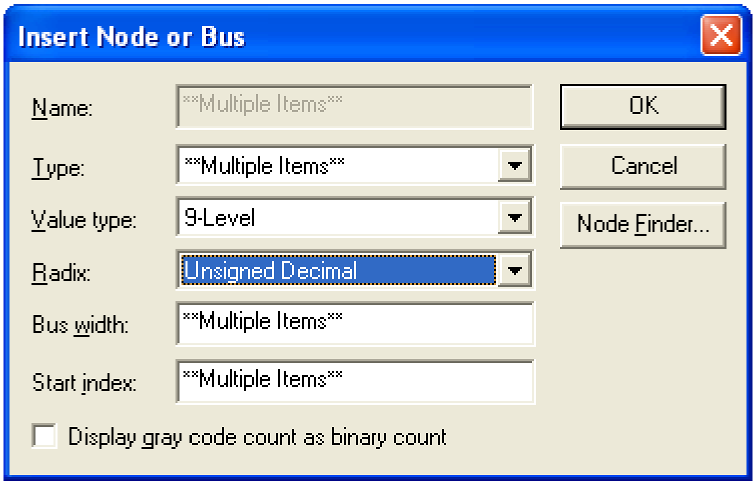
Nodes Found:

Name	Assignments	Type
A	Unassigned	Input Group
A[0]	Unassigned	Input
A[1]	Unassigned	Input
A[2]	Unassigned	Input
A[3]	Unassigned	Input
B	Unassigned	Input Group
B[0]	Unassigned	Input
B[1]	Unassigned	Input
B[2]	Unassigned	Input
B[3]	Unassigned	Input
Cout	Unassigned	Output
Sel	Unassigned	Input
Sum	Unassigned	Output Gro...
Sum[0]	Unassigned	Output

Selected Nodes:

Name	Assignments	Type	C
|Lab2|A	Unassigned	Input Group	U
|Lab2|B	Unassigned	Input Group	U
|Lab2|Sel	Unassigned	Input	U
|Lab2|Sum	Unassigned	Output Gro...	U

4. In the **Insert Node or Bus** window, change **Radix** to **Unsigned Decimal**. This will make it easier for us to enter values and interpret the results.



Name: **Multiple Items**

Type: **Multiple Items**

Value type: 9-Level

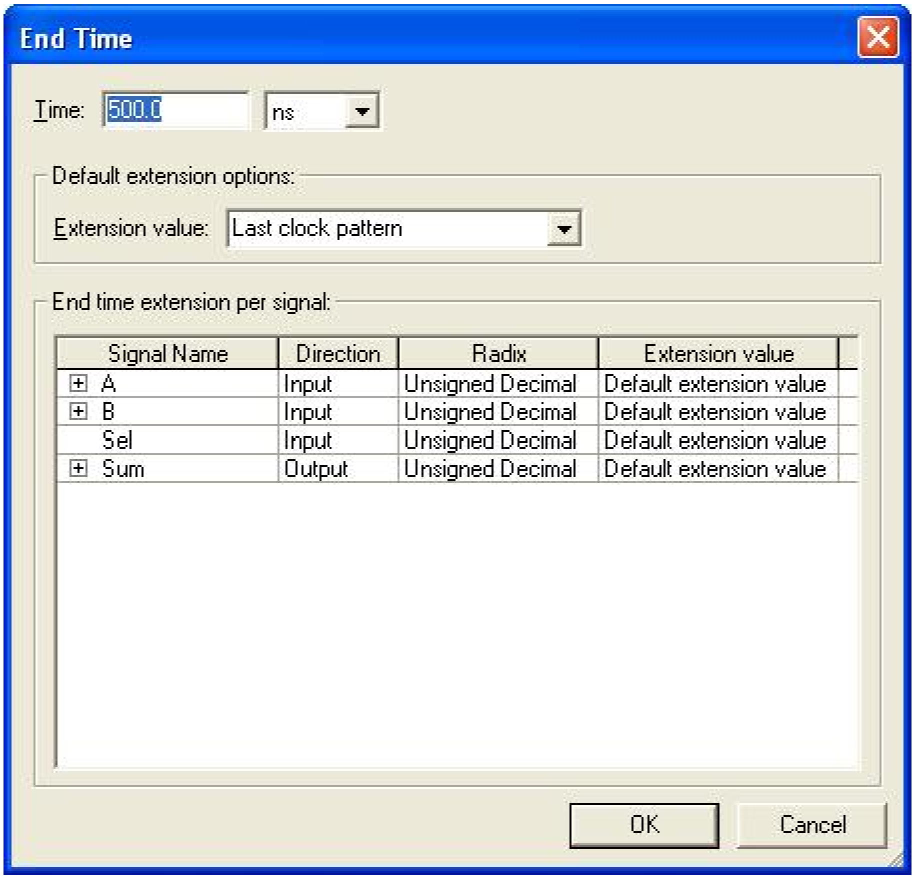
Radix: Unsigned Decimal

Bus width: **Multiple Items**

Start index: **Multiple Items**

☐ Display gray code count as binary count

5. Select **Edit → End Time...** to open the **End Time** window.
Change the end time to 500 ns. This is how long the simulated circuit will operate for.
Click **OK** to exit the **End Time** window.



Time: 500.0 ns

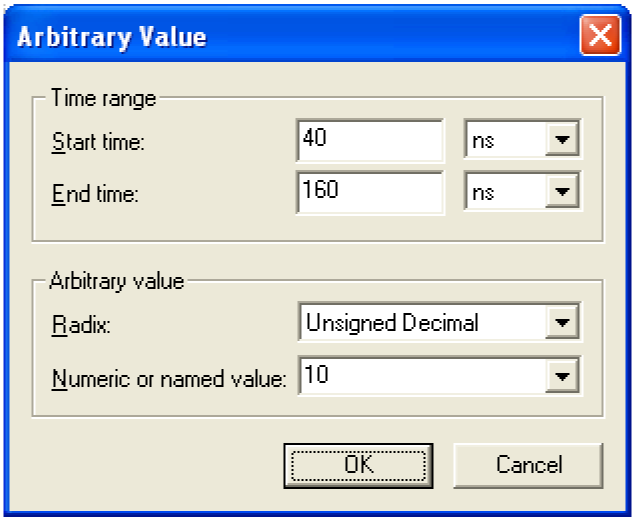
Default extension options:

Extension value: Last clock pattern

End time extension per signal:

Signal Name	Direction	Radix	Extension value
⊞ A	Input	Unsigned Decimal	Default extension value
⊞ B	Input	Unsigned Decimal	Default extension value
Sel	Input	Unsigned Decimal	Default extension value
⊞ Sum	Output	Unsigned Decimal	Default extension value

6. Click **A** in the **Name** column and press **Ctrl + Alt + B** to open the **Arbitrary Value** window.
End **40 ns** for the **Start time** and **160 ns** for the **End time**.
Enter **10** for the **Numeric or named value**.



Time range

Start time: 40 ns

End time: 160 ns

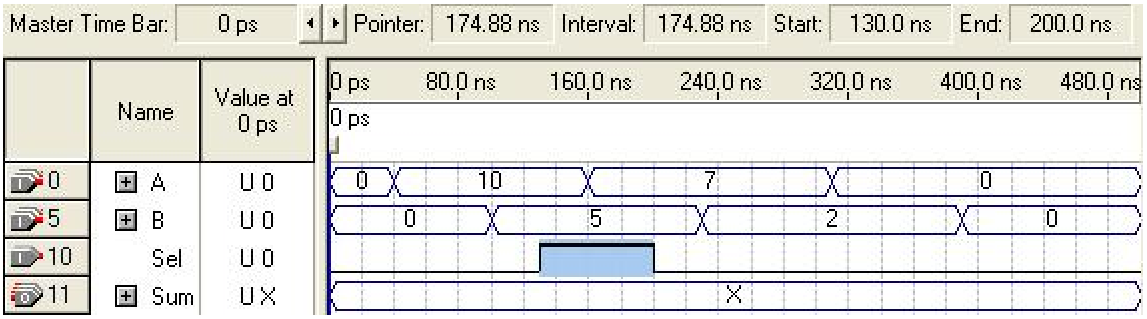
Arbitrary value

Radix: Unsigned Decimal

Numeric or named value: 10

Repeat this process for **A** to assign a value of **7** from **160 ns** to **310 ns**.
Assign **B** the value of **5** from **100 ns** to **230 ns**, and a value of **2** from **230 ns** to **390 ns**.

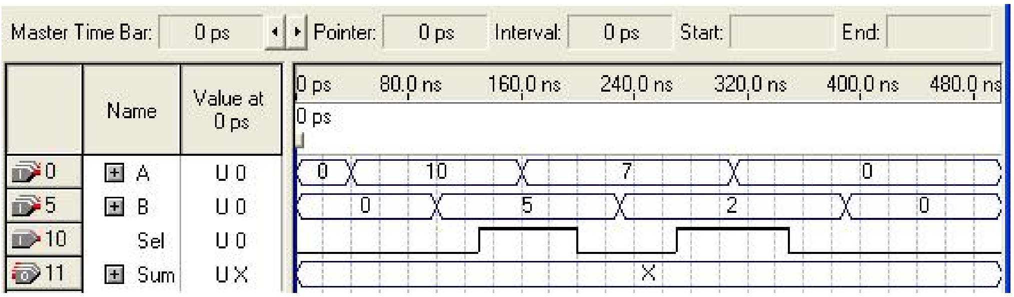
On the waveform row for **Sel**, click and drag to highlight the waveform from **130 ns** to **200 ns**. Press **Ctrl + Alt + 1** to assign this section of the waveform a value of **1**. Repeat this process for the value of **Sel** from **270 ns** to **350 ns**.



Master Time Bar: 0 ps Pointer: 174.88 ns Interval: 174.88 ns Start: 130.0 ns End: 200.0 ns

	Name	Value at 0 ps
0	A	U 0
5	B	U 0
10	Sel	U 0
11	Sum	U X

After completing the signal assignments the waveform graph should look similar to the image below:



Master Time Bar: 0 ps Pointer: 0 ps Interval: 0 ps Start: End:

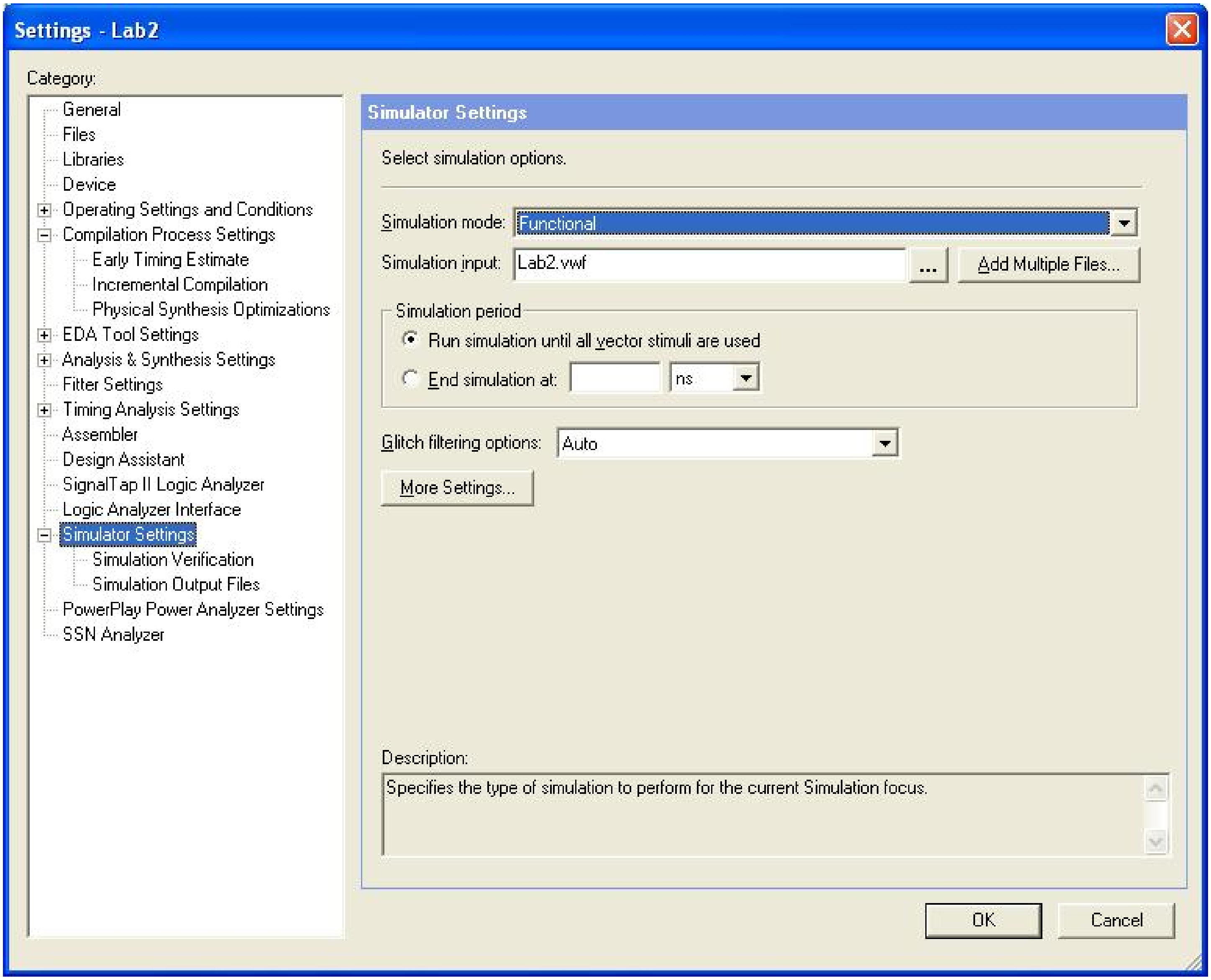
	Name	Value at 0 ps
0	A	U 0
5	B	U 0
10	Sel	U 0
11	Sum	U X

Leave **Sum** with a value of undefined (X), as values for this will be generated when the simulation runs.

Click the **Save** icon 💾 on the Menu bar and save the vwf file with the filename **Lab2.vwf**.

7. From the Toolbar, select **Assignments** → **Settings** to open the **Settings** window. Click the **Simulator Settings** heading and set **Simulation Mode** to **Functional**.

   Simulation input should be the name of your vwf file, **Lab2.vwf**. If not, click the ellipses ... and select **Lab2.vwf**.
   Click **OK** to save the changes and close the **Settings** window.



Settings - Lab2

Category:

- General
- Files
- Libraries
- Device
- ⊞ Operating Settings and Conditions
- ⊟ Compilation Process Settings
  - Early Timing Estimate
  - Incremental Compilation
  - Physical Synthesis Optimizations
- ⊞ EDA Tool Settings
- ⊞ Analysis & Synthesis Settings
- Fitter Settings
- ⊞ Timing Analysis Settings
- Assembler
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- ⊟ Simulator Settings
  - Simulation Verification
  - Simulation Output Files
- PowerPlay Power Analyzer Settings
- SSN Analyzer

Simulator Settings

Select simulation options.

Simulation mode: Functional

Simulation input: Lab2.vwf [...] [Add Multiple Files...]

Simulation period

● Run simulation until all vector stimuli are used

○ End simulation at: ns

Glitch filtering options: Auto

[More Settings...]

Description:

Specifies the type of simulation to perform for the current Simulation focus.

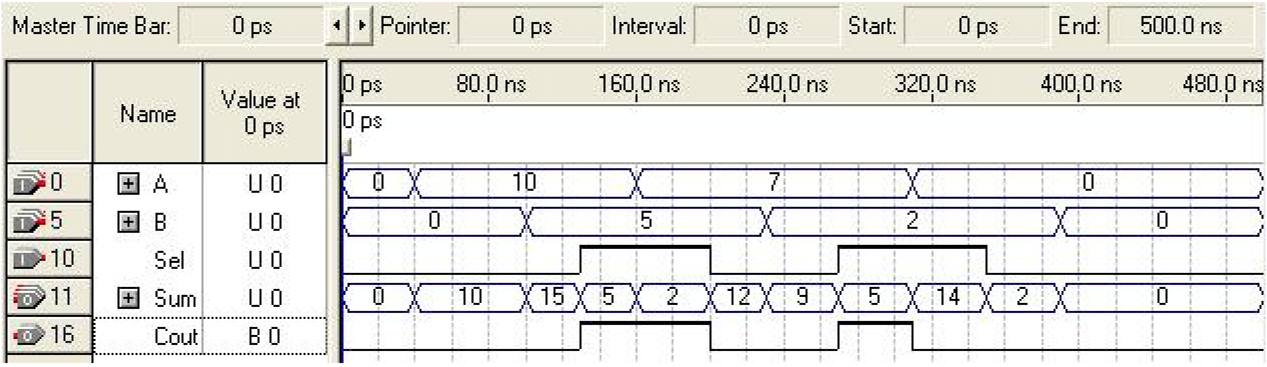
[OK] [Cancel]

8. Click the **Start Compilation** button ▶ on the Menu bar to compile the design. Ignore any warnings generated.

9. Select **Processing** → **Generate Functional Simulation Netlist** and click **OK** when it has completed.

10. Run the simulation by clicking the **Start Simulation** button ▶ on the Toolbar. Click **OK** once the simulator has completed.

11. The output waveforms for **Cout** and **Sum** should look similar to the screen capture below;

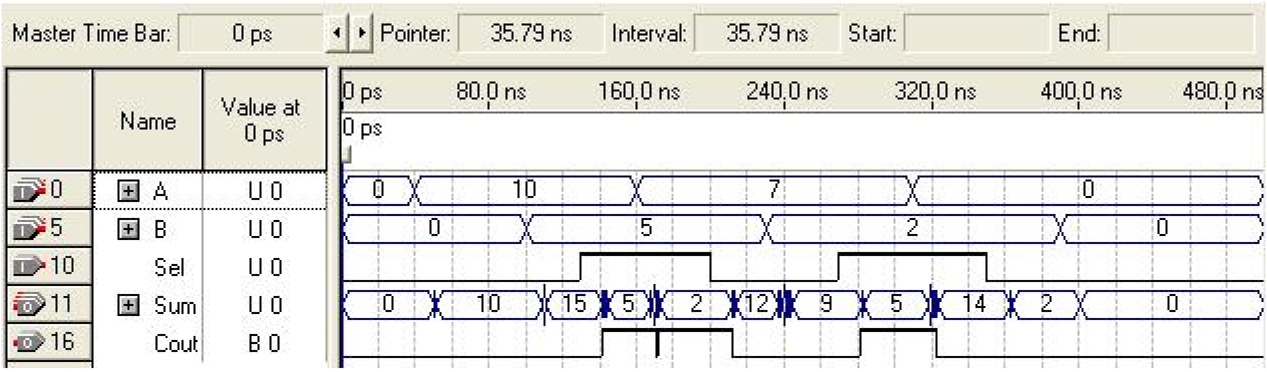


Master Time Bar: 0 ps | Pointer: 0 ps | Interval: 0 ps | Start: 0 ps | End: 500.0 ns

When **Sel** is low, **A** is added to **B** and the result is displayed as the value of **Sum**.
When **Sel** is high, **B** is subtracted from **A** and **Sum** is the result.

12. Change the simulation mode to **Timing** by selecting **Assignments → Settings** to open the **Settings** window.
Click the **Simulator Settings** heading and set **Simulation Mode** to **Timing**.

13. Click the **Start Simulation** button and observe the change in the output waveforms for **Cout** and **Sum**. In Timing mode the simulator uses estimated or actual timing information to test the logical operation and the timing of your design.

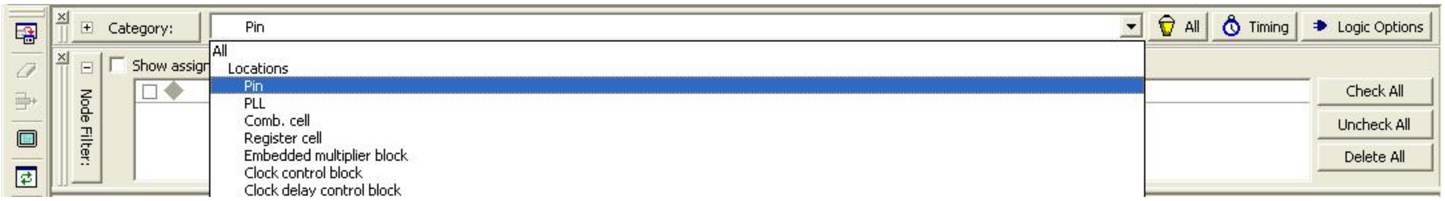


Master Time Bar: 0 ps | Pointer: 35.79 ns | Interval: 35.79 ns | Start: | End:

Task 4: Implementing the Design on the DE2 Board

1. From the Menu Bar select: **Assignments → Assignment Editor**

2. From the drop-down menu at the top, for **Category**, select **Pin**.



Category: Pin

3. Double-click **<<new>>** in the **To** column and select or type **A[0]**.
   Double-click the adjacent cell under **Location** and type **PIN_A13**. For a shortcut, you can simply type **A13** and the **Assignment Editor** will find **PIN_A13** for you.
   Continue to assign the pins as seen in the table below.

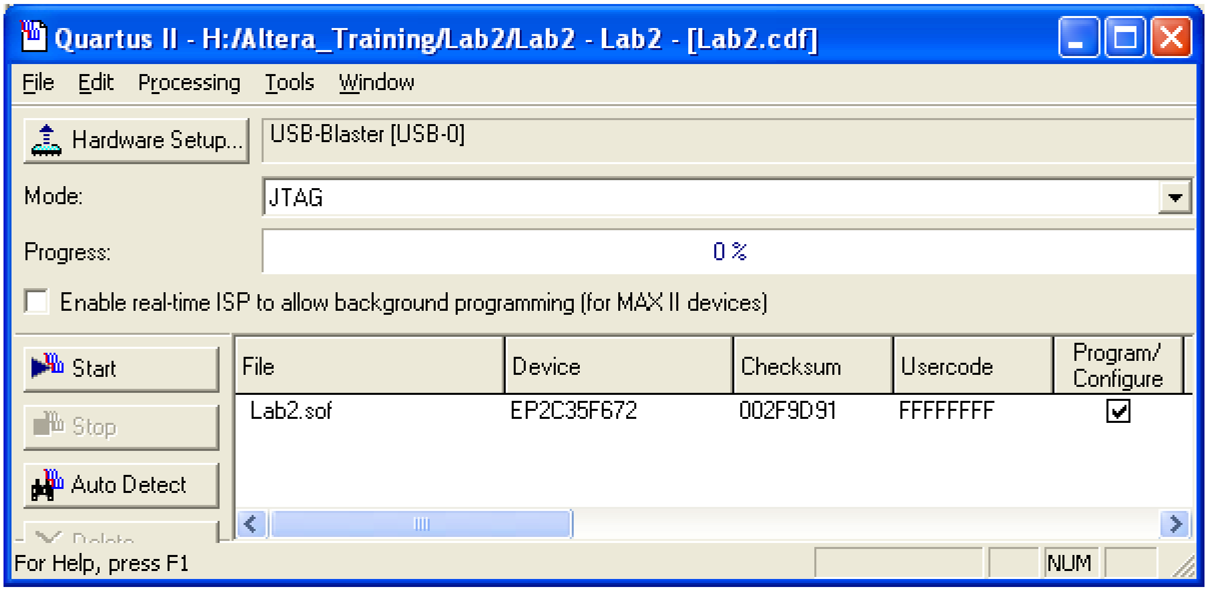
	To	Location	DE2 Board Description
1	A[0]	PIN_A13	SW9
2	A[1]	PIN_N1	SW10
3	A[2]	PIN_P1	SW11
4	A[3]	PIN_P2	SW12
5	B[0]	PIN_U3	SW14
6	B[1]	PIN_U4	SW15
7	B[2]	PIN_V1	SW16
8	B[3]	PIN_V2	SW17
9	Sel	PIN_N25	SW1
10	Sum[0]	PIN_AE22	LEDG0
11	Sum[1]	PIN_AF22	LEDG1
12	Sum[2]	PIN_W19	LEDG2
13	Sum[3]	PIN_V18	LEDG3
14	Cout	PIN_U18	LEDG4

Note: A complete list of pin assignments for the DE2 Development Board can be found here:
http://www.terasic.com.tw/attachment/archive/30/DE2_Pin_Table.pdf

4. Save the pin assignments by selecting **File → Save** from the Menu Bar, or by clicking the Save button 💾 on the toolbar.

5. Plug in and power on the DE2 board. Make sure that the **RUN/PROG Switch for JTAG/AS Modes** is in the **RUN** position.

6. In the Quartus II window click the **Programmer** button on the Toolbar to open the Programmer window
The **Hardware Setup…** must be **USB-Blaster [USB-0]**. If not, click the **Hardware Setup…** button and select **USB-Blaster [USB-0]** from the drop-down menu for **Currently selected hardware.**
Mode should be set to **JTAG**.
Make sure that the **File** is **Lab2.sof**, **Device** is **EP2C35F672**, and the **Program/Configure** box is checked.



Quartus II - H:/Altera_Training/Lab2/Lab2 - Lab2 - [Lab2.cdf]

Hardware Setup... USB-Blaster [USB-0]

Mode: JTAG

Progress: 0 %

□ Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure
Lab2.sof	EP2C35F672	002F9D91	FFFFFFFF	☑

For Help, press F1

Then click the **Start** button to program the DE2 board.
When the progress bar reaches 100%, programming is complete.

7. You can now test the program on the DE2 board by using the toggle switches located along the bottom of the board.
SW0 is the add/subtract selector. Low is for addition.
SW9 through SW12 are the four bits of data for A.
SW13 through SW17 are the four bits of data for B.
The output of the operation is displayed on the first four green LEDs located above the blue push buttons on the DE2 board.
LEDG4 is the indicator for Cout.

Lab 3

Advanced VHDL

Lab 3 –Advanced VHDL

This lab will demonstrate many advanced VHDL techniques and how they can be used to your advantage to create efficient VHDL code. Topics include operator balancing, resource sharing, preventing unwanted latches, and state machine encoding schemes.

Task 1: Create a New Project

1. **Start the New Project Wizard**
   Create a new project using the same procedure from the previous labs.
   Set the **Working Directory**, **Project Name**, and **Top-Level Design Entity** as seen in the table below:

Working Directory	H:\Altera_Training\Lab3
Project Name	Lab3a
Top-Level Design Entity	Lab3a

   *Note: After clicking **Next**, a window may pop up stating that the chosen working directory does not exist. Click **Yes** to create it.*
   Continue With the New Project Wizard using the same settings as before.

2. **Create a New File**
   Create a new VHDL file, as discussed previously.
   Copy and paste the following code into your new vhdl file and save it as Lab3a.vhd:

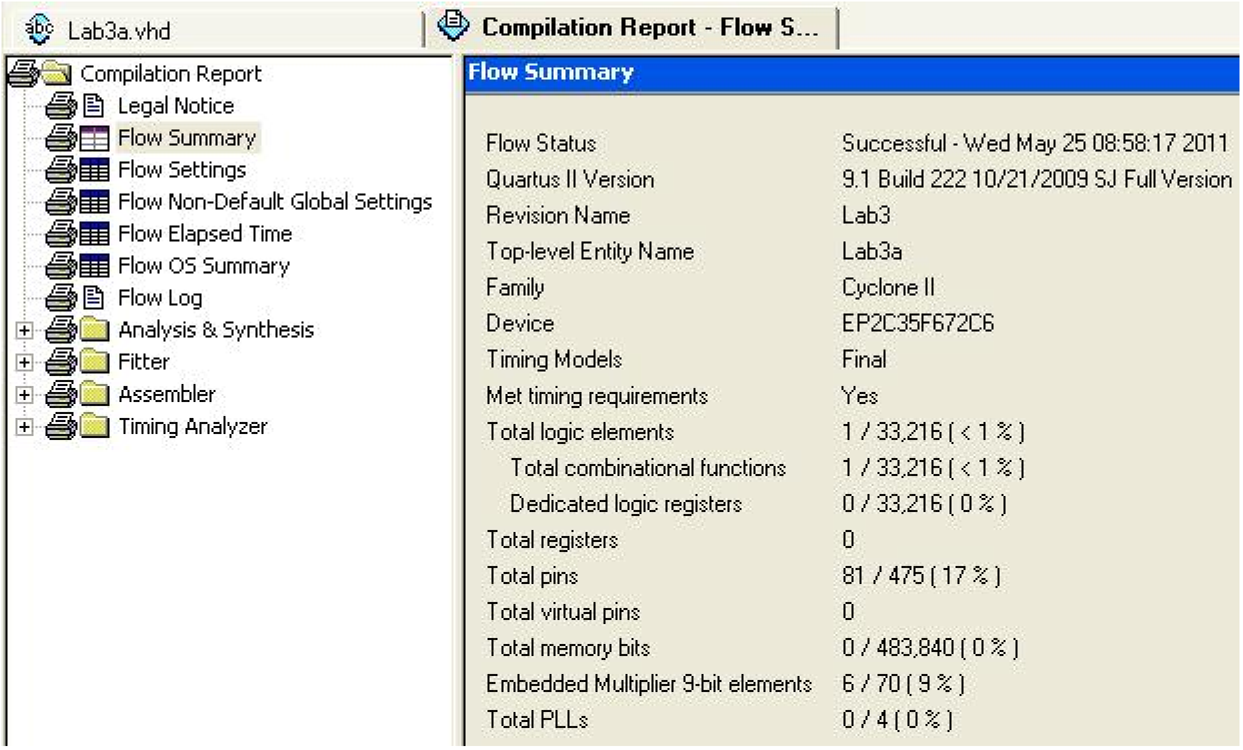
Lab 3a – Advanced VHDL – Resource Sharing Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY Lab3a IS
      PORT (w, x, y, z: IN std_logic_vector(7 DOWNTO 0);
                        result1, result2 : OUT std_logic_vector(23 DOWNTO 0);
                        CLK: IN std_logic);
END;

ARCHITECTURE Lab3a_arch OF Lab3a IS

      SIGNAL w_reg, x_reg, y_reg, z_reg: std_logic_vector(7 DOWNTO 0);
      SIGNAL temp_w_reg, temp_z_reg: std_logic_vector(7 DOWNTO 0);
      SIGNAL temp_xy_reg: std_logic_vector(15 DOWNTO 0);

BEGIN
      Mult_Process: PROCESS (clk)
      BEGIN
            IF RISING_EDGE(clk) THEN
                  w_reg <= w;
                  x_reg <= x;
                  y_reg <= y;
                  z_reg <= z;
                  result1 <= w_reg * x_reg * y_reg;
                  result2 <= x_reg * y_reg * z_reg;
--                temp_w_reg <= w_reg;
--                temp_z_reg <= z_reg;
--                temp_xy_reg <= x_reg * y_reg;
--                result1 <= temp_w_reg * temp_xy_reg;
--                result2 <= temp_xy_reg * temp_z_reg;
            END IF;
END PROCESS;

END Lab3a_arch;
```

Task 2: Operator Balancing and Resource Sharing

1. Compile that code as-is using the **Start Compilation** button ▶
   Ignore any warnings at this time.

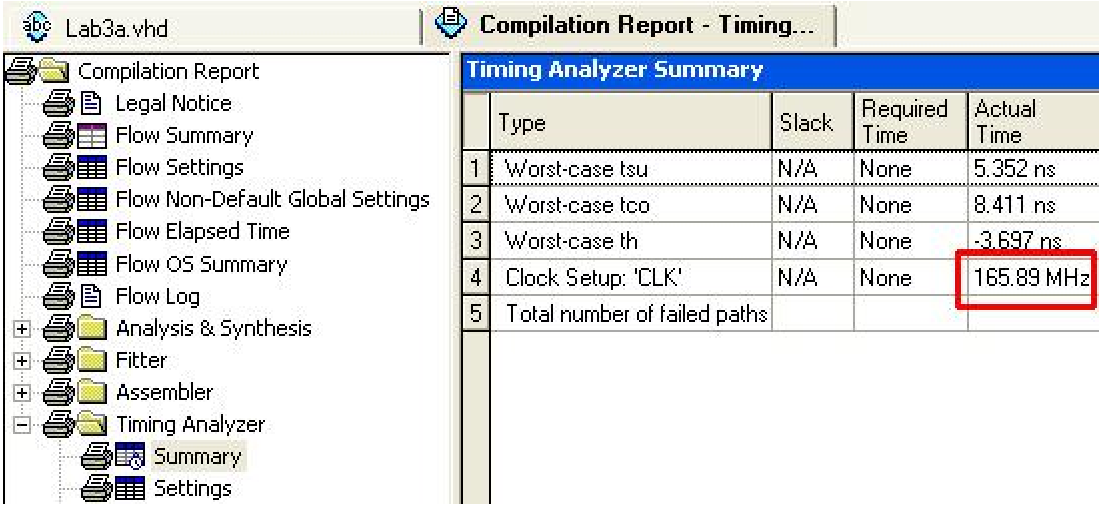
2. After compilation is complete, observe the **Flow Summary** in the **Compilation Report** that has been generated. Note the **Total logic elements**, **Total combinational functions**, **Dedicated logic registers**, and **Embedded Multiplier 9-bit elements** used in this design.



Flow Summary

Flow Status	Successful - Wed May 25 08:58:17 2011
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	Lab3
Top-level Entity Name	Lab3a
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	1 / 33,216 (< 1 %)
Total combinational functions	1 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	81 / 475 (17 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	6 / 70 (9 %)
Total PLLs	0 / 4 (0 %)

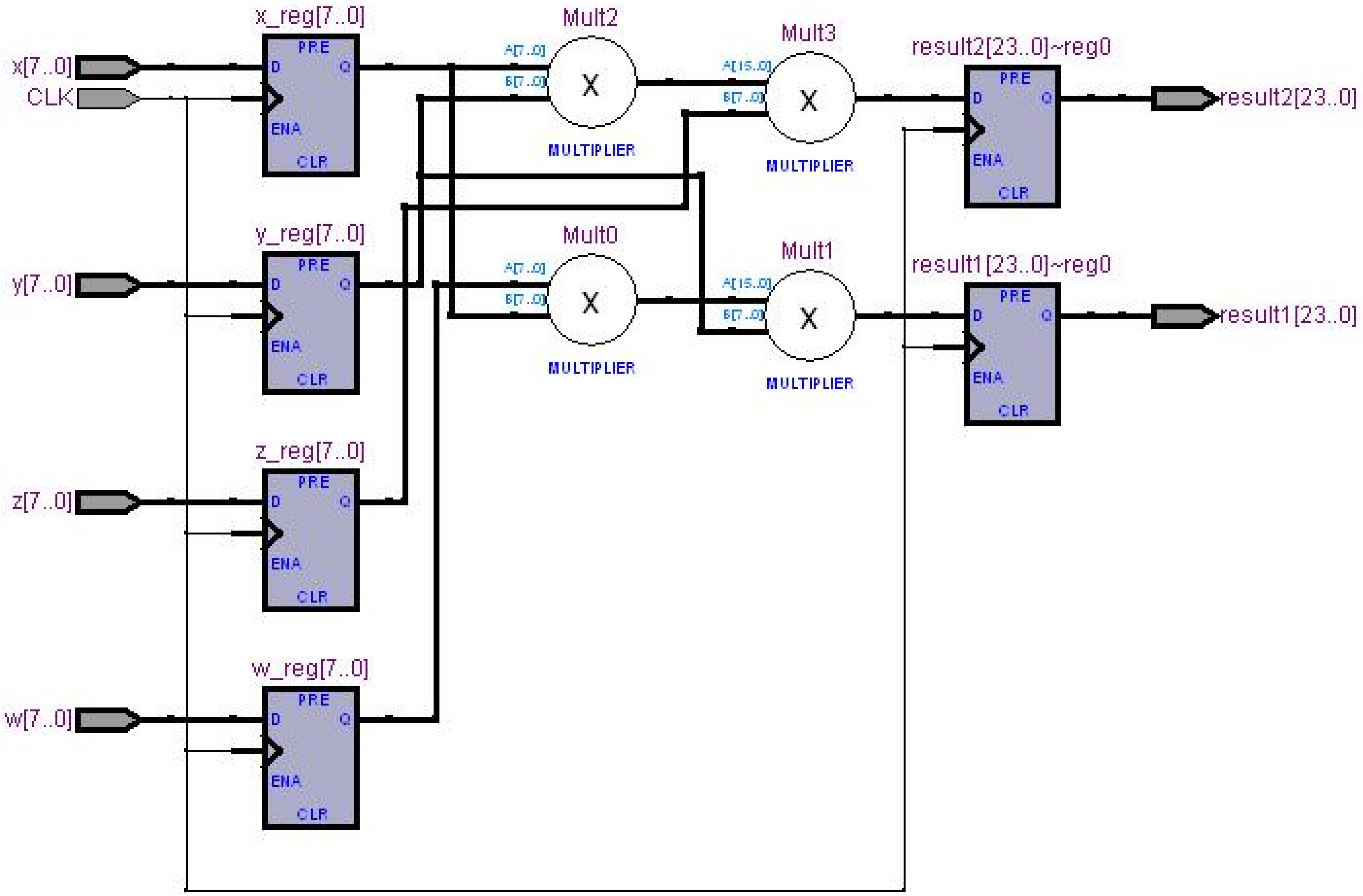
3. In the **Compilation Report**, click on **Timing Analyzer** to display the **Timing Analyzer Summary**. Here you can see that the maximum clock speed of this design is currently 165.89 MHz.



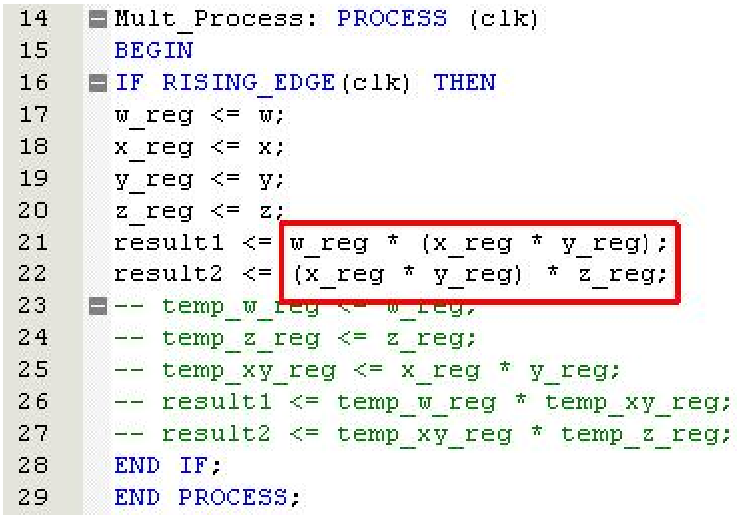
Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	5.352 ns
2	Worst-case tco	N/A	None	8.411 ns
3	Worst-case th	N/A	None	-3.697 ns
4	Clock Setup: 'CLK'	N/A	None	165.89 MHz
5	Total number of failed paths			

4. Select **Tools → Netlist Viewers → RTL Viewer** to open the **RTL Viewer**. This displays a schematic diagram of the circuit design. Take note of the number of multipliers used.

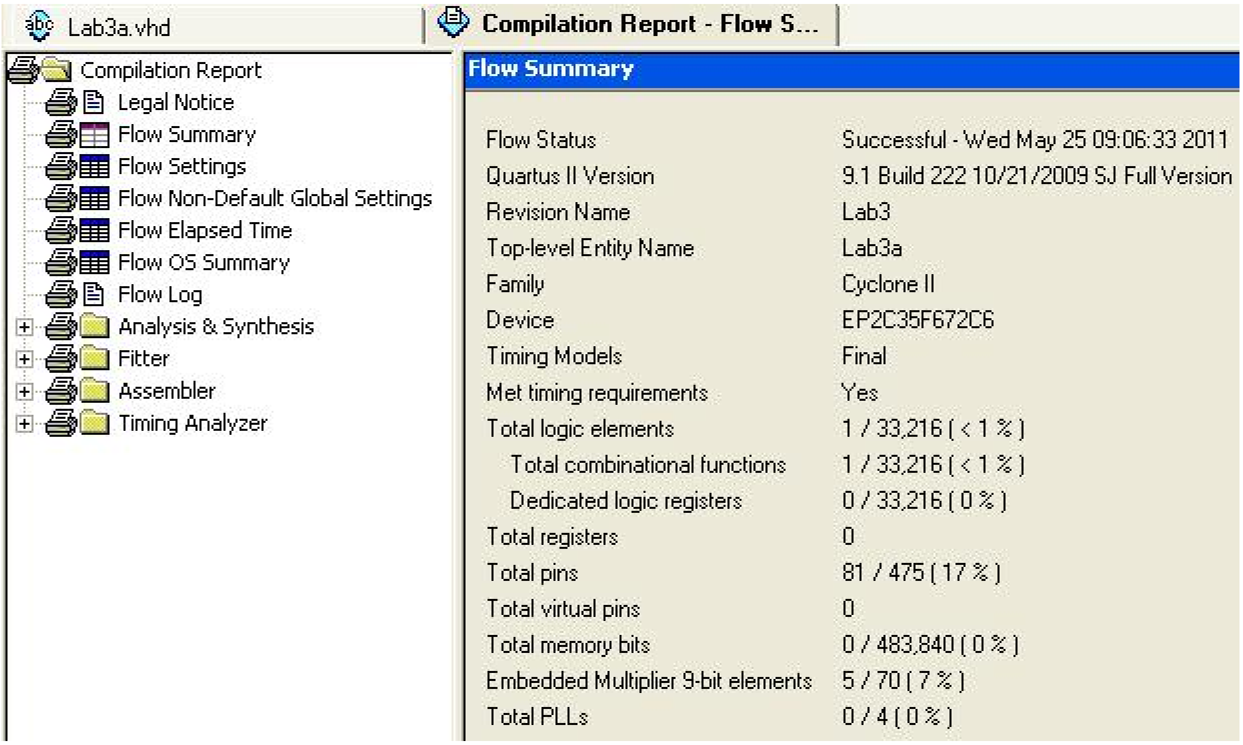


5. Go to **Lines 21** and **22** in the VHDL code and add parenthesis around **x_reg * y_reg** in each line. This optimizes resource sharing by not having to do the same operation multiple times.



```
14  Mult_Process: PROCESS (clk)
15  BEGIN
16  IF RISING_EDGE(clk) THEN
17  w_reg <= w;
18  x_reg <= x;
19  y_reg <= y;
20  z_reg <= z;
21  result1 <= w_reg * (x_reg * y_reg);
22  result2 <= (x_reg * y_reg) * z_reg;
23  -- temp_w_reg <= w_reg;
24  -- temp_z_reg <= z_reg;
25  -- temp_xy_reg <= x_reg * y_reg;
26  -- result1 <= temp_w_reg * temp_xy_reg;
27  -- result2 <= temp_xy_reg * temp_z_reg;
28  END IF;
29  END PROCESS;
```

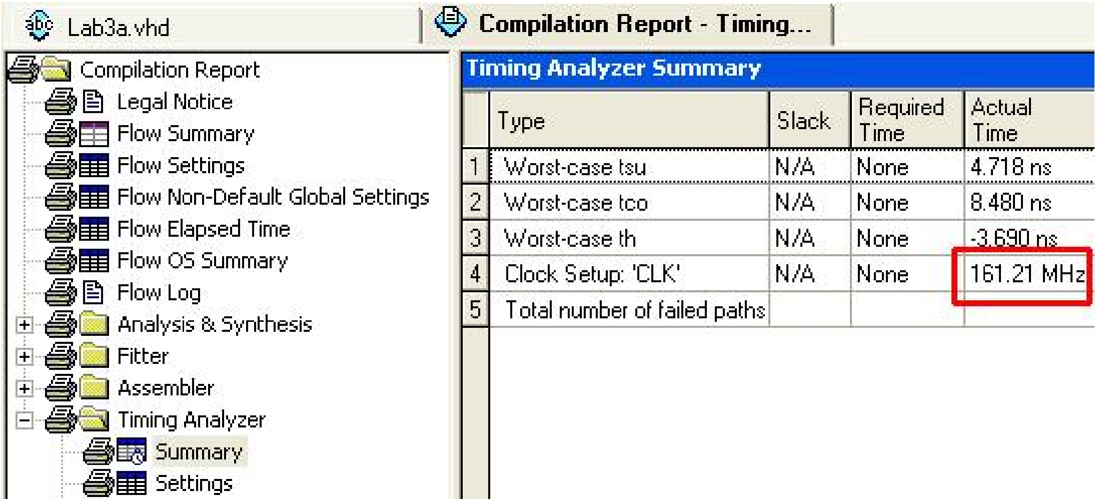
6. Save the changes to **Lab3a.vhd**, then compile the design again and view the results in the **Compilation Report**. Ignore any warnings at this time.
   This time it can be seen that the number of **Embedded Multiplier 9-bit elements** has been reduced by one, this is due to the resource sharing technique we used.



Flow Summary

Flow Status	Successful - Wed May 25 09:06:33 2011
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	Lab3
Top-level Entity Name	Lab3a
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	1 / 33,216 (< 1 %)
Total combinational functions	1 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	81 / 475 (17 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	5 / 70 (7 %)
Total PLLs	0 / 4 (0 %)

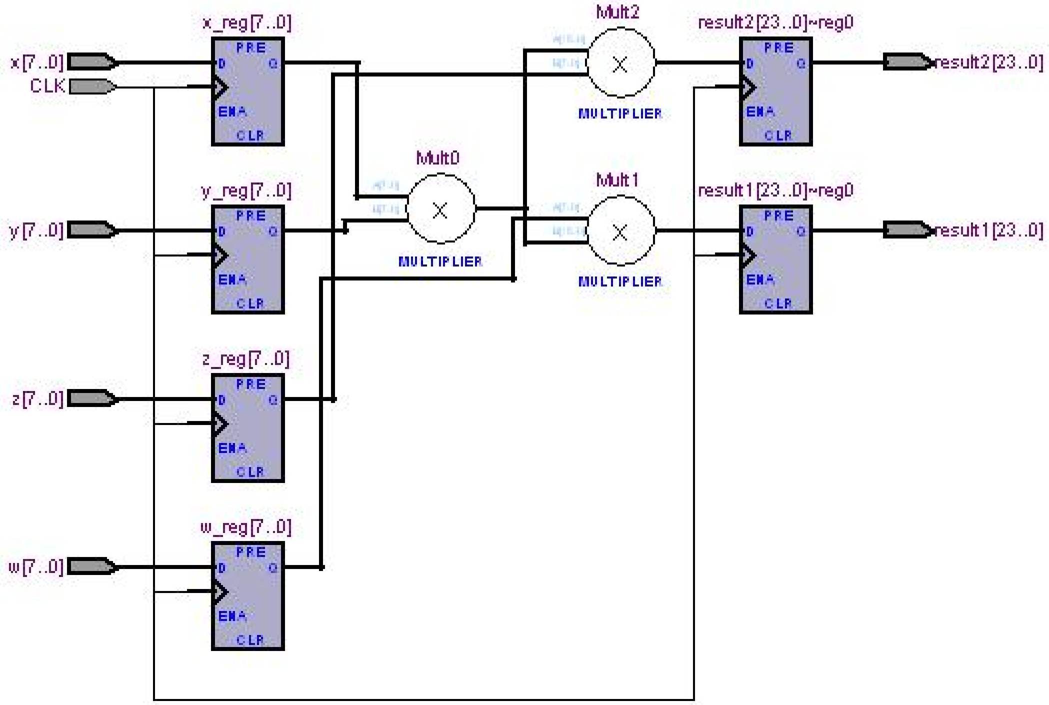
7. View the **Timing Analyzer Summary** to see that although fewer components have been utilized, the clock speed has been reduced slightly.



Timing Analyzer Summary

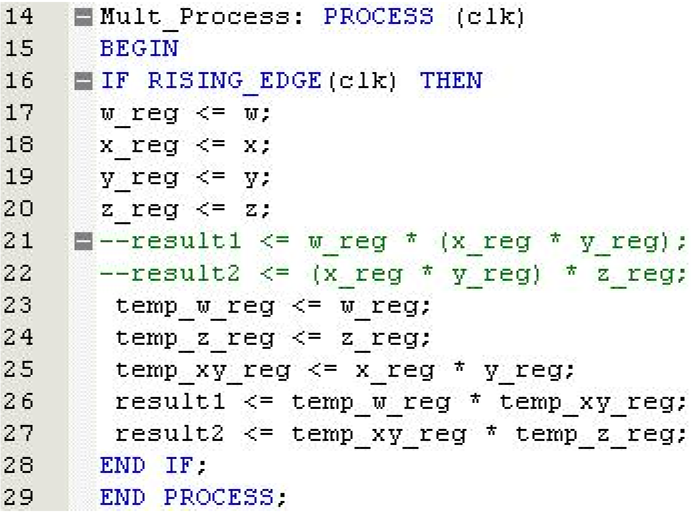
	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	4.718 ns
2	Worst-case tco	N/A	None	8.480 ns
3	Worst-case th	N/A	None	-3.690 ns
4	Clock Setup: 'CLK'	N/A	None	161.21 MHz
5	Total number of failed paths			

8. Open the **RTL Viewer** again to see the new schematic diagram for the circuit design. Compare this schematic diagram to the previous one and you can see that the number of multipliers used has been reduced by one.



9. Go to **Lines 21** and **22** again and highlight both lines by clicking and dragging with the mouse. Then click the **Comment selected text** button on the vertical Text Editor Menu Bar

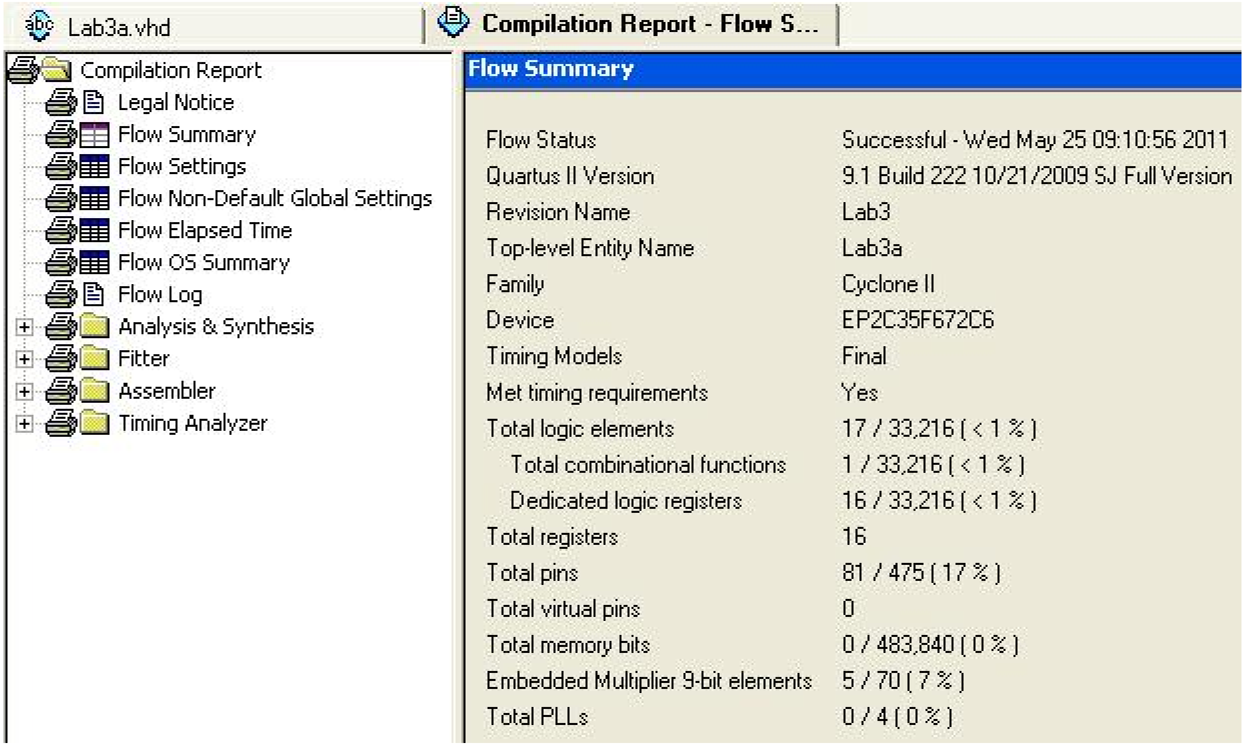
Next, highlight **Lines 23** though **27** and click the **Uncomment selected text** button which is located below the **Comment selected text** button. The section of code should now look similar to the image below:



```
14  Mult_Process: PROCESS (clk)
15  BEGIN
16  IF RISING_EDGE(clk) THEN
17  w_reg <= w;
18  x_reg <= x;
19  y_reg <= y;
20  z_reg <= z;
21  --result1 <= w_reg * (x_reg * y_reg);
22  --result2 <= (x_reg * y_reg) * z_reg;
23   temp_w_reg <= w_reg;
24   temp_z_reg <= z_reg;
25   temp_xy_reg <= x_reg * y_reg;
26   result1 <= temp_w_reg * temp_xy_reg;
27   result2 <= temp_xy_reg * temp_z_reg;
28  END IF;
29  END PROCESS;
```

The newly uncommented code uses temporary registers to increase clocking speed by pipelining.

10. Save the changes to **Lab3a.vhd**, then compile the design again and view the results in the **Compilation Report**. Ignore any warnings at this time.

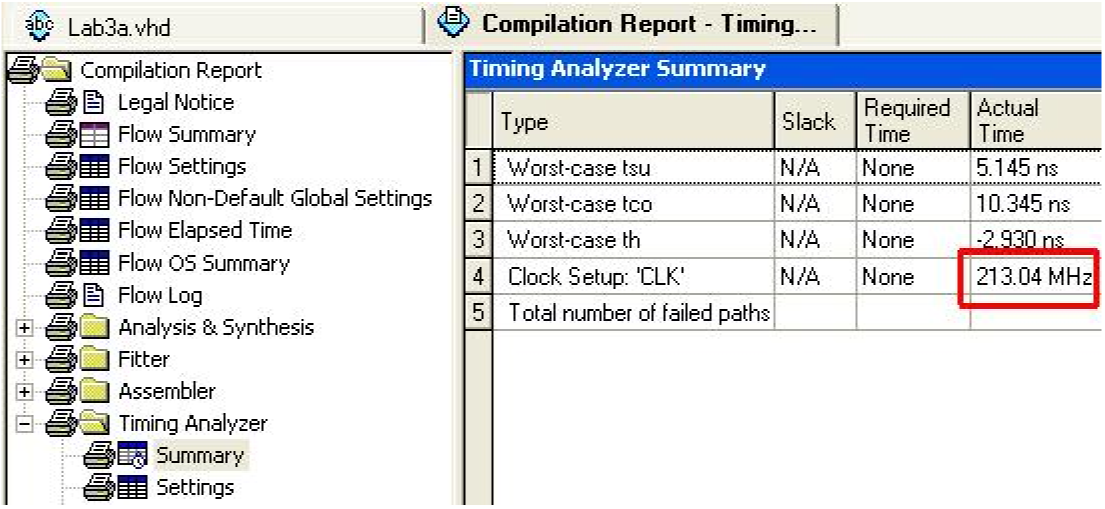


Flow Summary

Flow Status	Successful - Wed May 25 09:10:56 2011
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	Lab3
Top-level Entity Name	Lab3a
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	17 / 33,216 (< 1 %)
Total combinational functions	1 / 33,216 (< 1 %)
Dedicated logic registers	16 / 33,216 (< 1 %)
Total registers	16
Total pins	81 / 475 (17 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	5 / 70 (7 %)
Total PLLs	0 / 4 (0 %)

The **Compilation Report** indicates that many more logic elements and registers are being used in the design now.

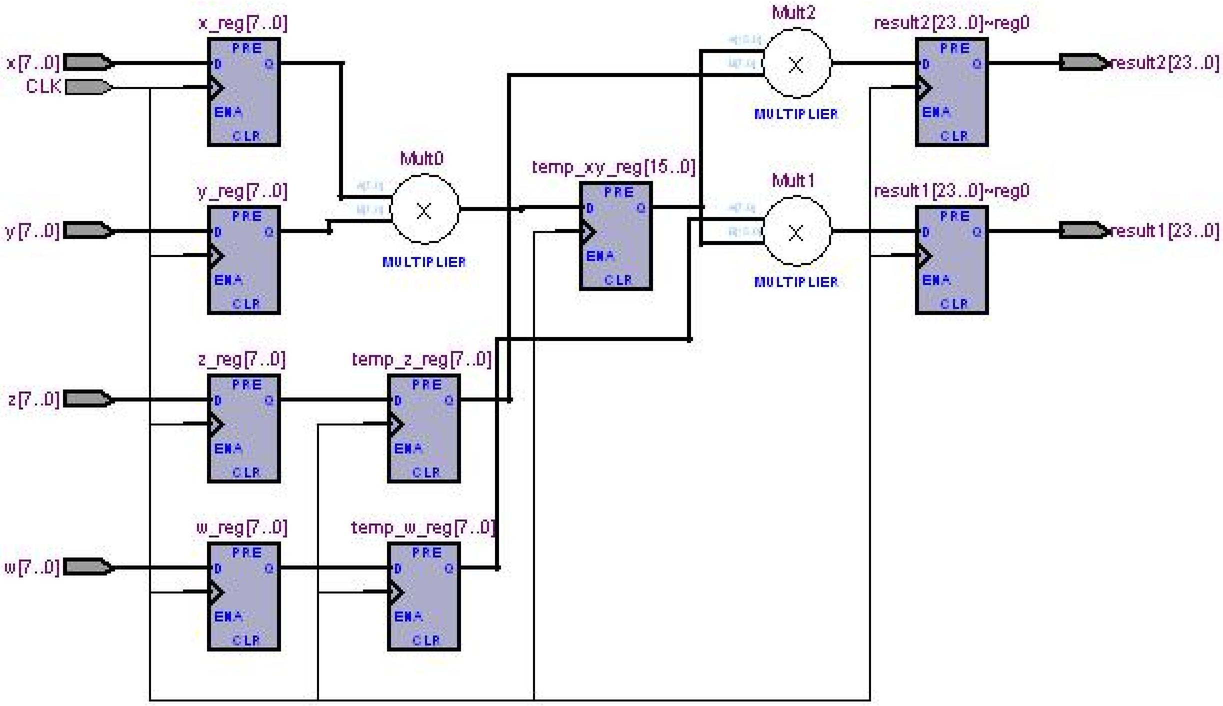
11. Open the **Timing Analyzer Summary** to see that the clock speed has increased due to the pipelining.



Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	5.145 ns
2	Worst-case tco	N/A	None	10.345 ns
3	Worst-case th	N/A	None	-2.930 ns
4	Clock Setup: 'CLK'	N/A	None	213.04 MHz
5	Total number of failed paths			

12. Open the **RTL Viewer** again to see the new schematic diagram. Compared to the previous schematic diagrams, you can see where the additional registers have been placed.



Task 3: Preventing Latches

1. **Create a New File**
   Create a new VHDL file, as discussed previously.
   Copy and paste the following code into your new vhdl file and save it as **Lab3b.vhd**:

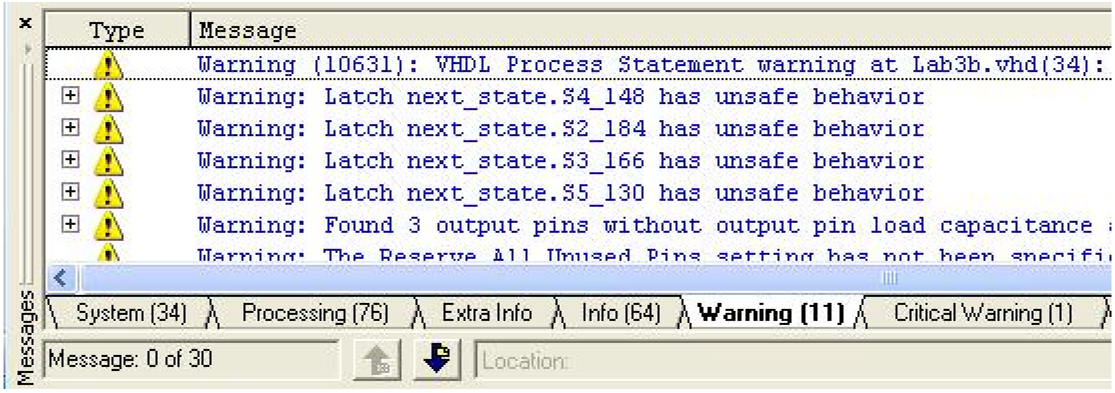
Lab 3b – Advanced VHDL – State Machine Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Lab3b IS
  PORT (A, B, aclr: IN std_logic;
        CLK: IN std_logic;
        state_data: OUT std_logic_vector(2 DOWNTO 0));
END;

ARCHITECTURE Lab3b_arch OF Lab3b IS

  TYPE State_types IS (S1, S2, S3, S4, S5);

  ATTRIBUTE syn_encoding: STRING;
  ATTRIBUTE syn_encoding OF State_types : TYPE IS
--     "00001 00010 00100 01000 10000";    -- One-Hot Encoding
--     "000 001 010 100 110";              -- Minimum Bits
--     "000 001 011 010 110";              -- Gray Encoding
--     "000 100 101 111 011";              -- Johnson Encoding
       "000 001 010 011 100";              -- Sequential Encoding

    SIGNAL current_state, next_state: State_types;

BEGIN
      state_transitions: PROCESS (aclr, clk)
      BEGIN
            IF aclr = '1' THEN
                  current_state <= S1;
            ELSIF rising_edge (clk) THEN
                  current_state <= next_state;
            END IF;
      END PROCESS state_transitions;

  State_machine:PROCESS (A, B, current_state, next_state)
  BEGIN
--      next_state <= current_state;
      CASE current_state IS
        WHEN S1 =>
          IF A='1' THEN
            next_state <= S2;
          END IF;
        WHEN S2 =>
          IF A='0' THEN
            next_state <= S1;
          ELSIF A='1' THEN
            next_state <= S3;
          END IF;
        WHEN S3 =>
          IF A='0' THEN
            next_state <= S1;
          ELSIF A='1' AND B='1' THEN
            next_state <= S4;
          END IF;
        WHEN S4 =>
          IF B='1' THEN
            next_state <= S5;
          ELSE
            next_state <= S1;
          END IF;
        WHEN S5 =>
          IF (A='0' AND B='1') THEN
            next_state <= S2;
          ELSIF (A='1' AND B='0') THEN
            next_state <= S3;
          END IF;
        WHEN OTHERS=>
          next_state <= current_state;
      END CASE;
  END PROCESS;

  Output_Process: PROCESS (current_state)
  BEGIN
            state_data <= (OTHERS => '0');
            CASE current_state IS
                  WHEN S1 =>
                        state_data <= "001";
                  WHEN S2 =>
                        state_data <= "010";
                  WHEN S3 =>
                        state_data <= "011";
                  WHEN S4 =>
                        state_data <= "100";
                  WHEN S5 =>
                        state_data <= "101";
                  WHEN OTHERS =>
                        NULL;
            END CASE;
  END PROCESS;

END Lab3b_arch;
```

2. Select **Project → Set as Top-Level-Entity** to specify that **Lab3b.vhd** is now the main file for the design.

   Compile that code using the **Start Compilation** button ▶
   Click **OK** when compilation is complete.

3. Click on the **Warning** tab in the **Messages** pane to view the warnings.

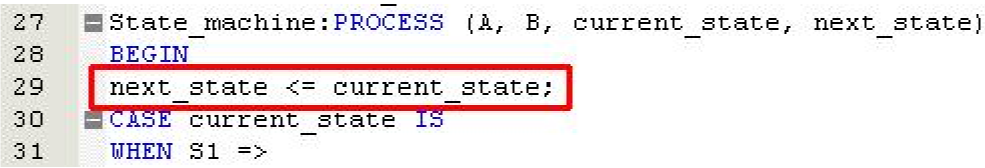


```
Type   Message
       Warning (10631): VHDL Process Statement warning at Lab3b.vhd(34):
       Warning: Latch next_state.S4_148 has unsafe behavior
       Warning: Latch next_state.S2_184 has unsafe behavior
       Warning: Latch next_state.S3_166 has unsafe behavior
       Warning: Latch next_state.S5_130 has unsafe behavior
       Warning: Found 3 output pins without output pin load capacitance
       Warning: The Reserve All Unused Pins setting has not been specifi
```

Warning (10631), **Warning: Latch next_state.S4_148 has unsafe behavior**, and others are due to unintentional latching conditions caused by not initializing the signals assigned within a case statement.

4. Open the **RTL Viewer** to see the complex logic that this unintentional latching creates. Upon simulation the circuit design may also produce unintended results due to these latches.

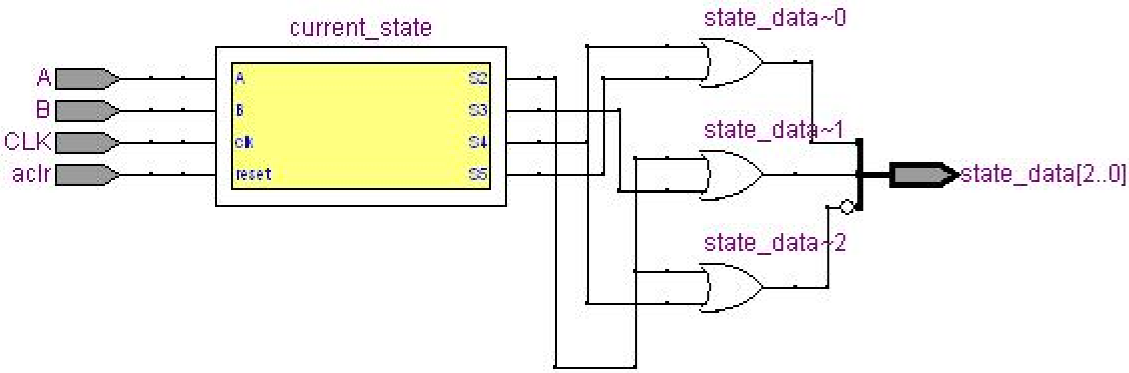
5. Highlight **Line 29** in the Lab3b.vhd code and click the **Uncomment selected text** button
This adds an initialization value at the beginning of the case statement for the `next_state` signal.



```
27  State_machine:PROCESS (A, B, current_state, next_state)
28  BEGIN
29  next_state <= current_state;
30  CASE current_state IS
31  WHEN S1 =>
```

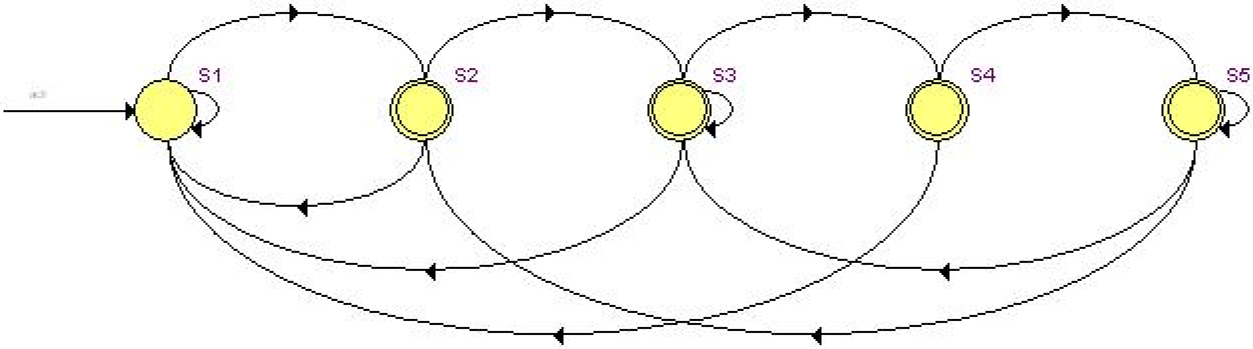
6. Save the changes and compile the design. Now it can be seen that the only warnings left relate to specifics regarding pin assignments on the physical FPGA device.

7. Open the **RTL Viewer** again to see that now the logic is much simplified, and what we intended.

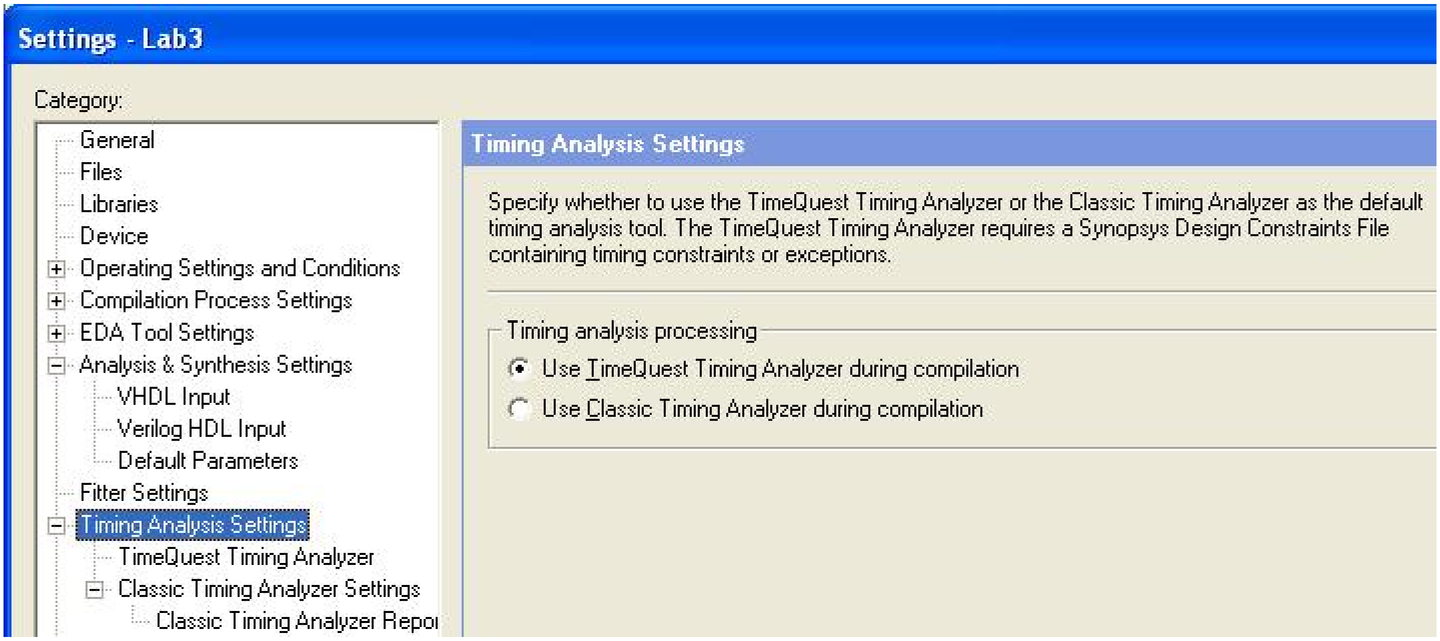


Task 4: State Machine Encoding

1. Double-clicking the yellow **current_state** box in the **RTL Viewer** will open up the **State Machine Viewer**. This graphically shows the state transitions and their associated conditions. Alternately, the **State Machine Viewer** can be opened by selecting **Tools → Netlist Viewers → State Machine Viewer**.



2. Go to **Assignments → Settings** and select **Timing Analysis Settings**. Select **Use TimeQuest Timing Analyzer during compilation** under the **Timing analysis processing** heading. Click **OK** to close the **Settings** window, then compile the design again.



Settings - Lab3

Category:

General
Files
Libraries
Device
Operating Settings and Conditions
Compilation Process Settings
EDA Tool Settings
Analysis & Synthesis Settings
VHDL Input
Verilog HDL Input
Default Parameters
Fitter Settings
Timing Analysis Settings
TimeQuest Timing Analyzer
Classic Timing Analyzer Settings
Classic Timing Analyzer Repo

Timing Analysis Settings

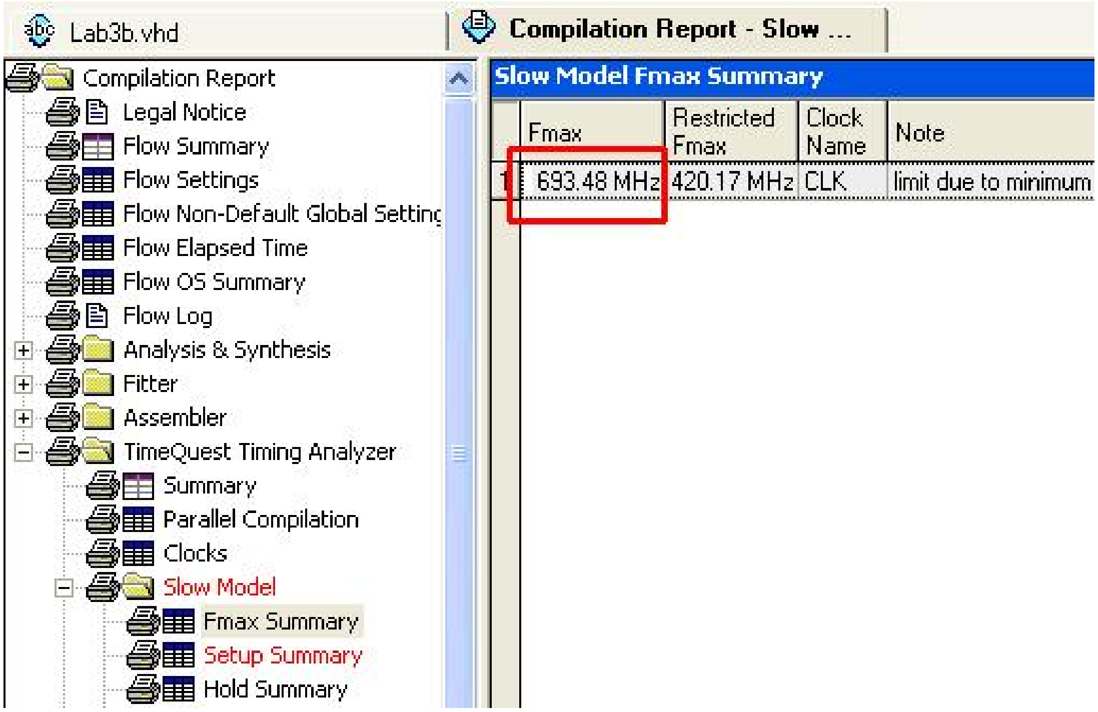
Specify whether to use the TimeQuest Timing Analyzer or the Classic Timing Analyzer as the default timing analysis tool. The TimeQuest Timing Analyzer requires a Synopsys Design Constraints File containing timing constraints or exceptions.

Timing analysis processing

● Use TimeQuest Timing Analyzer during compilation

○ Use Classic Timing Analyzer during compilation

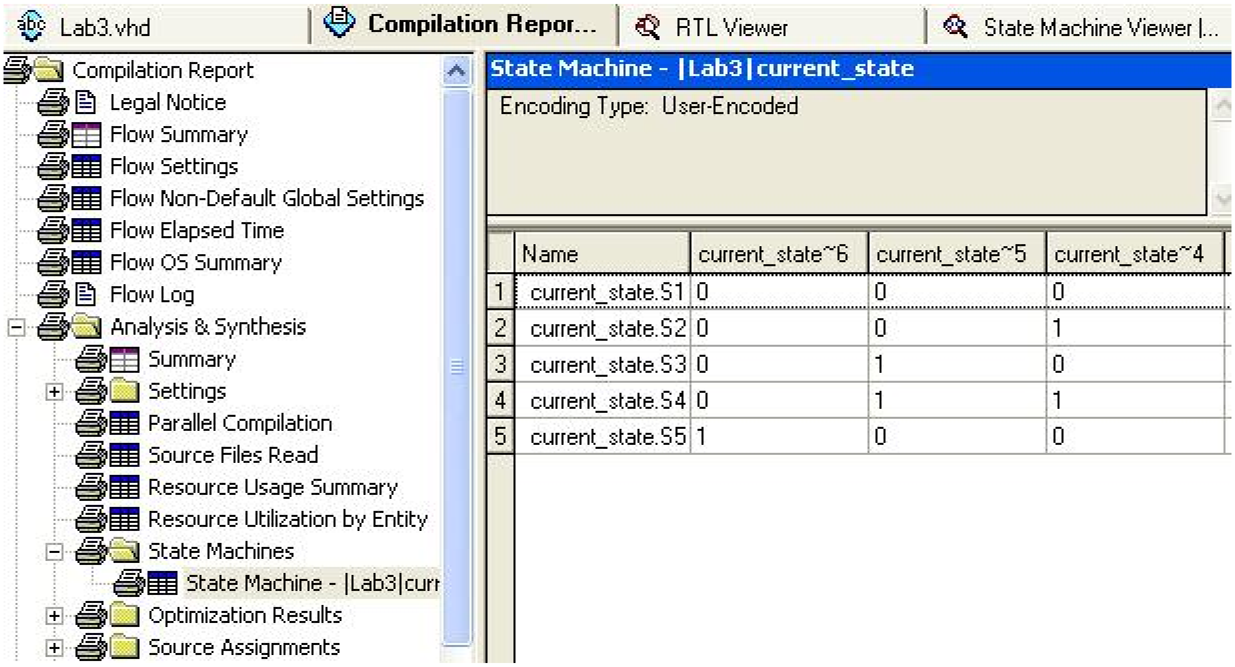
3. In the **Compilation Report – Flow Summary**, note the number of **Total logic elements** and **Dedicated logic registers** used. Then under the **TimeQuest Timing Analyzer Summary**, select **Slow Model** and view the maximum clock frequency of the design.



Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	693.48 MHz	420.17 MHz	CLK	limit due to minimum

4. In the **Compilation Report**, click on **Analysis & Synthesis** and then expand the subheading **State Machines**. This shows the encoding type used by the compiler to identify the different states. The encoding type is currently **User-Encoded** meaning that the VHDL code specifies the encoding used. Alternately the compiler can be configured to use a specific encoding type. The VHDL code currently is using a **Sequential Encoding** scheme to identify the states.

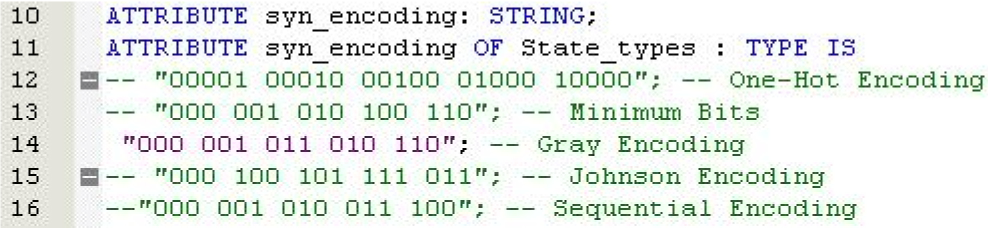


State Machine - |Lab3|current_state

Encoding Type: User-Encoded

	Name	current_state~6	current_state~5	current_state~4
1	current_state.S1	0	0	0
2	current_state.S2	0	0	1
3	current_state.S3	0	1	0
4	current_state.S4	0	1	1
5	current_state.S5	1	0	0

5. Select **Line 16** in the **Lab3b.vhd** code and click the **Comment selected text** button. Then select **Line 14** and click the **Uncomment selected text** button. This changes the encoding type to **Gray Encoding**. Save the changes to Lab3b.vhd and compile the design to see the results of changing the state machine encoding.

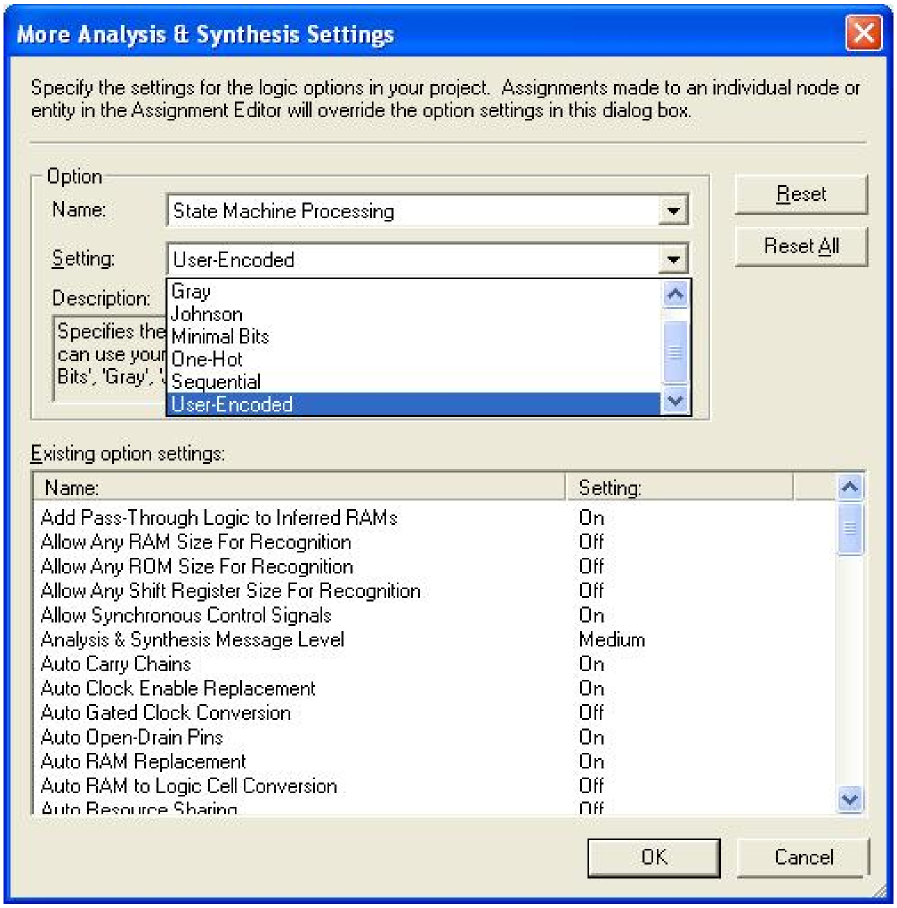


```
10    ATTRIBUTE syn_encoding: STRING;
11    ATTRIBUTE syn_encoding OF State_types : TYPE IS
12  ⊟-- "00001 00010 00100 01000 10000"; -- One-Hot Encoding
13    -- "000 001 010 100 110"; -- Minimum Bits
14     "000 001 011 010 110"; -- Gray Encoding
15  ⊟-- "000 100 101 111 011"; -- Johnson Encoding
16    --"000 001 010 011 100"; -- Sequential Encoding
```

6. Again look at the **Flow Summary** of the **Compilation Report**, and **Fmax Summary** under the **Slow Model** in the **TimQuest Timing Analyzer** section of the **Compilation Report**. Compare the current values to the previous results.

7. **Lines 12** through **16** in the **Lab3b.vhd** file each contain a different encoding type which can be selected by uncommenting the desired line and then compiling the design again and comparing the changes to the number of design components used and the clock speed.

8. Alternately, the encoding type can be set by the Quartus II software by selecting **Assignments → Settings** and then selecting the **Analysis & Synthesis** heading and clicking the **More Settings…** button to open the **More Analysis & Synthesis Settings** window. Then select **State Machine Processing** in the drop down menu for **Name**. In the drop down menu for **Setting** select the desired state machine encoding then click **OK** in the **More Analysis & Synthesis Settings** window and in the **Settings** window to save the changes.



More Analysis & Synthesis Settings

Specify the settings for the logic options in your project. Assignments made to an individual node or entity in the Assignment Editor will override the option settings in this dialog box.

Option

Name: State Machine Processing

Setting: User-Encoded

Description:

Specifies the
can use your
Bits', 'Gray', '

Existing option settings:

Name:	Setting:
Add Pass-Through Logic to Inferred RAMs	On
Allow Any RAM Size For Recognition	Off
Allow Any ROM Size For Recognition	Off
Allow Any Shift Register Size For Recognition	Off
Allow Synchronous Control Signals	On
Analysis & Synthesis Message Level	Medium
Auto Carry Chains	On
Auto Clock Enable Replacement	On
Auto Gated Clock Conversion	Off
Auto Open-Drain Pins	On
Auto RAM Replacement	On
Auto RAM to Logic Cell Conversion	Off
Auto Resource Sharing	Off

Lab 4

VHDL Testbenching

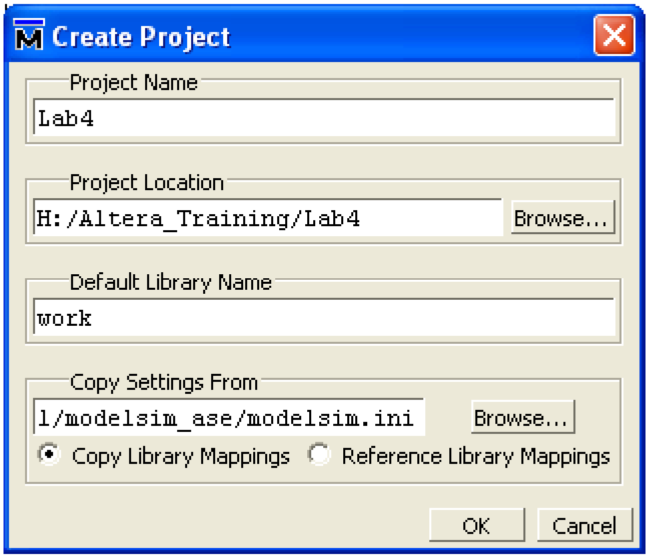
Lab 4 – VHDL Testbenching

This lab will show how to write testbench files to test your circuit designs using the ModelSim software.

Task 1: Create a New Project

1. **Start the ModelSim software.**
   From the Windows Start Menu, select:
   All Programs → Altera → ModelSim-Altera 6.5b (Quartus II 9.1) Starter Edition → ModelSim-Altera 6.5b (Quartus II 9.1) Starter Edition

2. Select **File → New → Project** to open the **Create Project** window.
   For the **Project Name** type **Lab4**
   For the **Project Location** type **H:/Altera_Training/Lab4**
   Then click **OK.**
   Click **OK** again if you are asked to create the project directory.



Project Name: Lab4

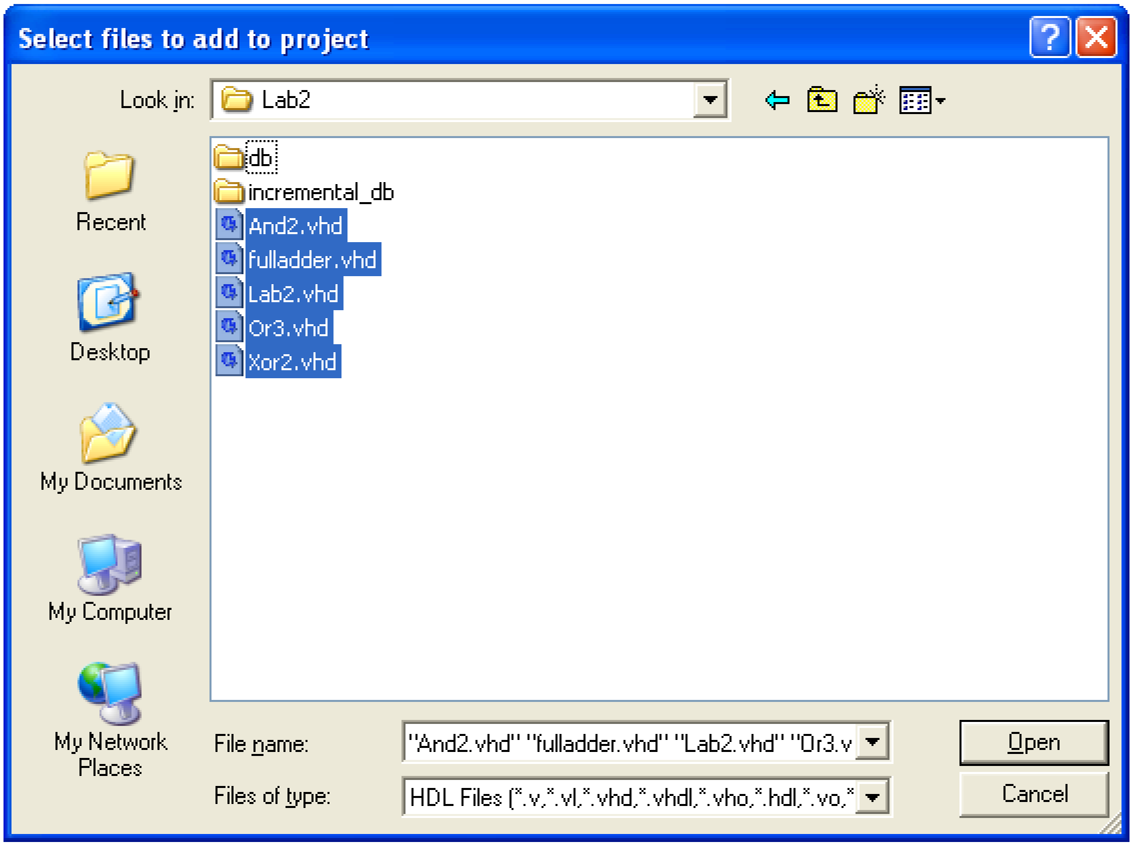
Project Location: H:/Altera_Training/Lab4

Default Library Name: work

Copy Settings From: l/modelsim_ase/modelsim.ini

● Copy Library Mappings ○ Reference Library Mappings

3. In the **Add items to the Project** window click **Add Existing File.**
   Click **Browse…** on the **Add file to Project** window and browse to **H:/Altera_Training/Lab2**
   Select all of the **.vhd** files and click open.



Look in: Lab2

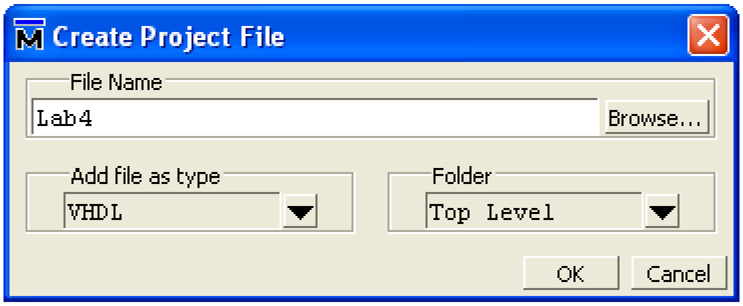
db
incremental_db
And2.vhd
fulladder.vhd
Lab2.vhd
Or3.vhd
Xor2.vhd

File name: "And2.vhd" "fulladder.vhd" "Lab2.vhd" "Or3.v

Files of type: HDL Files (*.v,*.vl,*.vhd,*.vhdl,*.vho,*.hdl,*.vo,*

Then click **OK** in the **Add file to Project** window.

4. Click **Create New File** in the **Add items to the Project** window.
   Type **Lab4** in the **Create Project File** window and click **OK.**



File Name: Lab4

Add file as type: VHDL

Folder: Top Level

Click **Close** in the **Add items to the Project** window to close it.

5. Double-click on **Lab4.vhd** in the project window of ModelSim to open an editor pane. Copy and paste the following code into the editor pane:

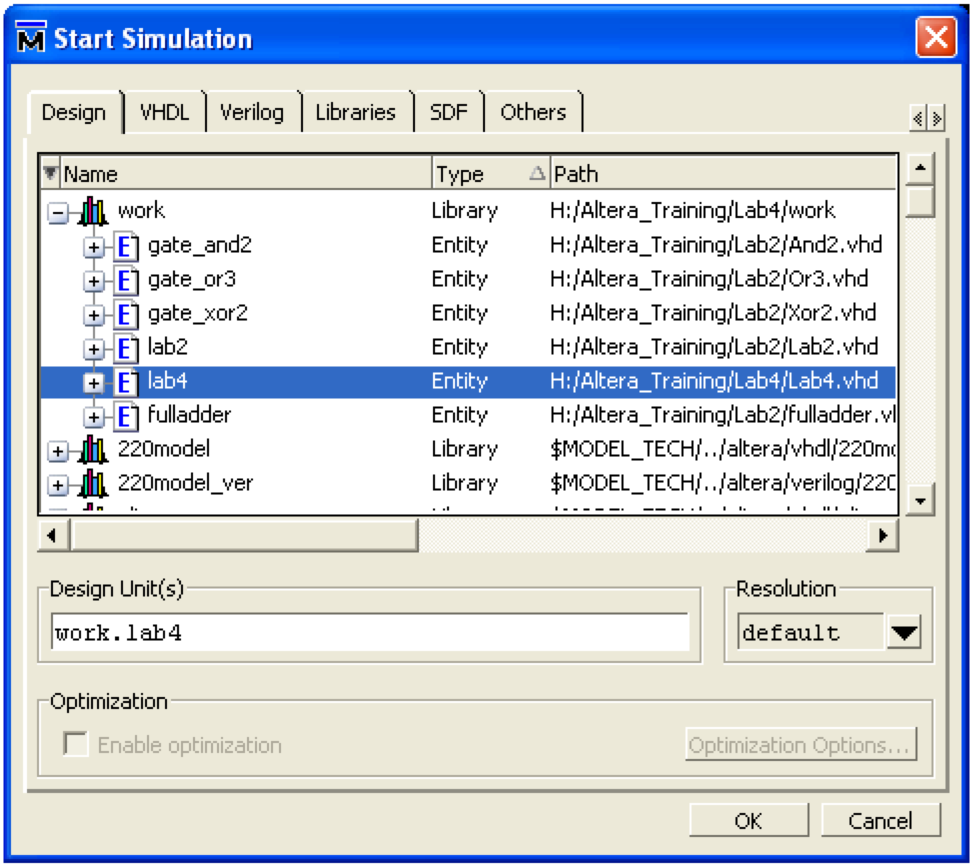
Lab 4 – Testbenching – Testbench Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Lab4 IS
END Lab4;

ARCHITECTURE TBarch OF Lab4 IS
  COMPONENT Lab2 IS
    PORT (A: IN std_logic_vector(3 DOWNTO 0);
          B: IN std_logic_vector(3 DOWNTO 0);
          Sel: IN std_logic;
          Sum: OUT std_logic_vector(3 DOWNTO 0);
          Cout: OUT std_logic);
  END COMPONENT;

SIGNAL A_s, B_s, Sum_s: std_logic_vector(3 DOWNTO 0);
SIGNAL Sel_s, Cout_s, clk: std_logic;

BEGIN
  CompToTest: Lab2
    PORT MAP (A_s, B_s, Sel_s, Sum_s, Cout_s);

  clk_Process: PROCESS
  BEGIN
    clk <= '0';
    WAIT FOR 10 ns;
    clk <= '1';
    WAIT FOR 10 ns;
  END PROCESS;

  Adder_Process: PROCESS
  BEGIN
    A_s <= "0000"; B_s <= "0000"; Sel_s <= '0';
    WAIT FOR 10 ns;
    A_s <= "0011"; B_s <= "0001"; Sel_s <= '1';
    WAIT FOR 10 ns;
    A_s <= "0100"; B_s <= "0010"; Sel_s <= '0';
    WAIT FOR 10 ns;
    WAIT;
  END PROCESS;
END TBarch;
```

6. Save the changes to **Lab4.vhd** by clicking the **Save** button 💾 on the toolbar.
   Compile all the files by selecting **Compile → Compile All** from the menu bar.

Task 2: Simulating the Circuit

1. Select **Simulate → Start Simulation** from the menu bar to open the **Start Simulation** window. Expand the **work** directory, select **lab4**, and click **OK**.



Name	Type	Path
work	Library	H:/Altera_Training/Lab4/work
gate_and2	Entity	H:/Altera_Training/Lab2/And2.vhd
gate_or3	Entity	H:/Altera_Training/Lab2/Or3.vhd
gate_xor2	Entity	H:/Altera_Training/Lab2/Xor2.vhd
lab2	Entity	H:/Altera_Training/Lab2/Lab2.vhd
lab4	Entity	H:/Altera_Training/Lab4/Lab4.vhd
fulladder	Entity	H:/Altera_Training/Lab2/fulladder.v
220model	Library	$MODEL_TECH/../altera/vhdl/220m
220model_ver	Library	$MODEL_TECH/../altera/verilog/22

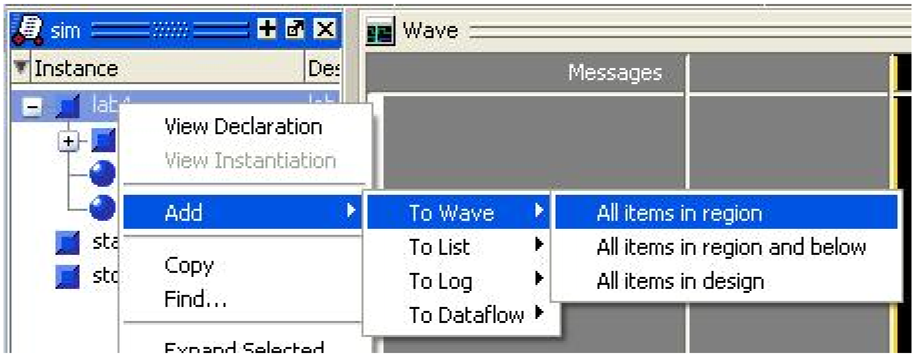
Design Unit(s): work.lab4

Resolution: default

Optimization

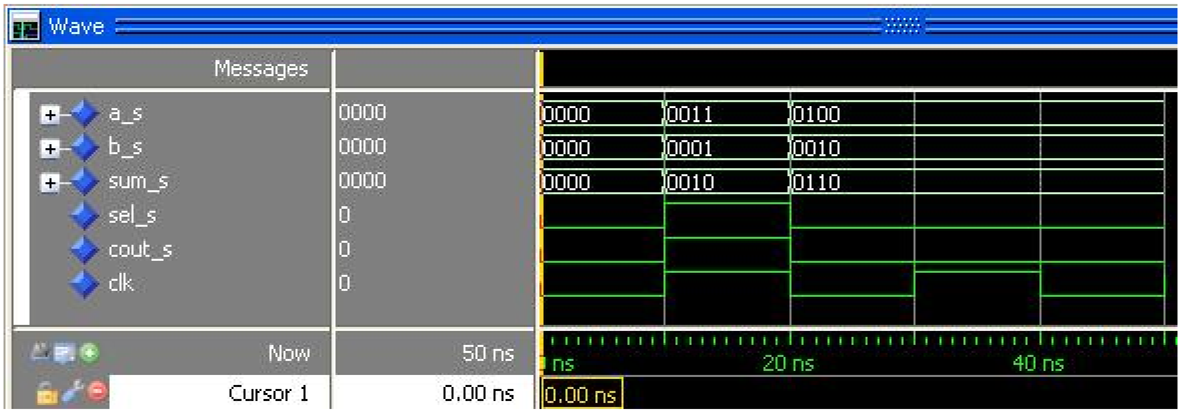
☐ Enable optimization

2. In the **sim** pane, right-click on **lab4** and select **Add → To wave → All items in region**.
This adds the signals from the test bench file to the waveform so we can see their values when the simulation runs. You may need to play around with the window sizing some in order to expand the waveform window to a reasonable size.



3. Change the simulation **Run Length** to **50 ns** by typing it into the entry box on the toolbar 50 ns Then click the **Run** button to start the simulation.

   Press the **minus key** - to zoom out to an appropriate size. To zoom in, press **shift +** +
   The resulting waveform should be similar to the image below:

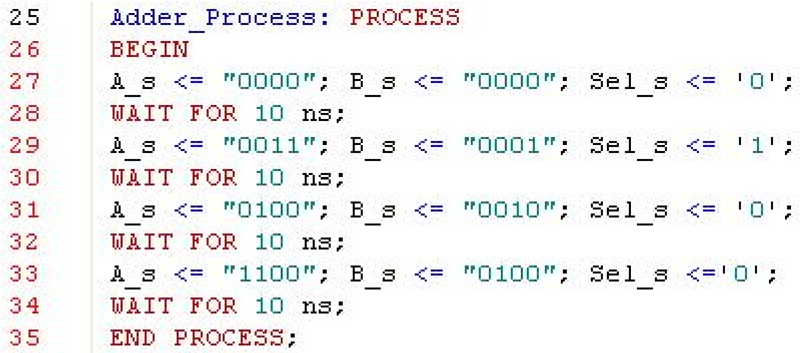


Task 3: Editing the Test Bench

1. Look at the **Adder_Process** of the **lab4.vhd** code, here is where the values for the two four-bit numbers are assigned, as well as where the add/subtract selection is made. More input combinations can be created to test additional input scenarios. To see what output an overflow condition would generate, add the following on **lines 33** and **34**:

   ```
   A_s <= "1100"; B_s <= "0100"; Sel_s <='0';
         WAIT FOR 10 ns;
   ```

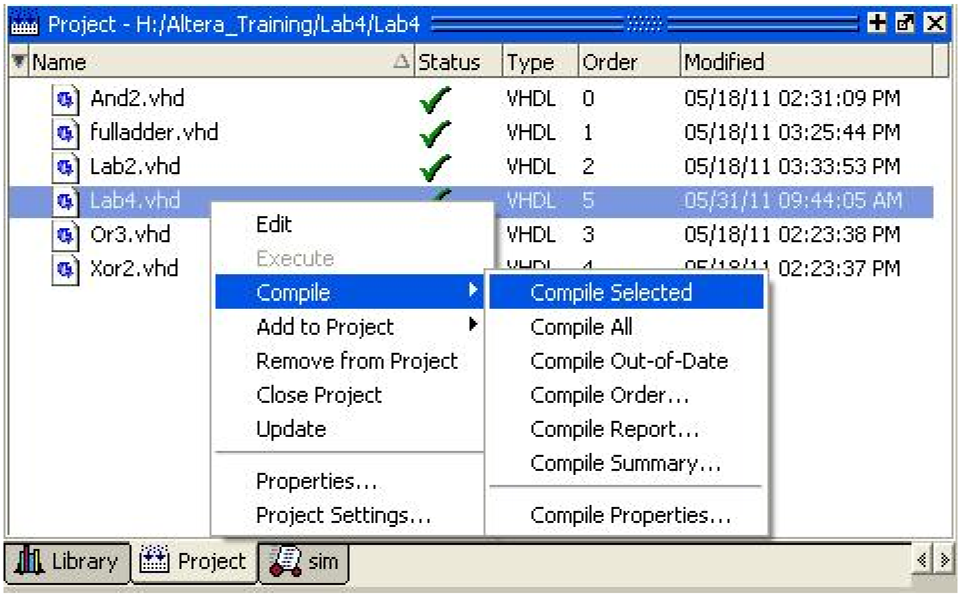
   The resulting Adder_Process should look similar to the image below:



   ```
   25  Adder_Process: PROCESS
   26  BEGIN
   27  A_s <= "0000"; B_s <= "0000"; Sel_s <= '0';
   28  WAIT FOR 10 ns;
   29  A_s <= "0011"; B_s <= "0001"; Sel_s <= '1';
   30  WAIT FOR 10 ns;
   31  A_s <= "0100"; B_s <= "0010"; Sel_s <= '0';
   32  WAIT FOR 10 ns;
   33  A_s <= "1100"; B_s <= "0100"; Sel_s <='0';
   34  WAIT FOR 10 ns;
   35  END PROCESS;
   ```

   This will cause our adder/subtractor to add **12** (1100) with **4** (0100) which would give us the result of **16** (10000), but because of the four-bit limit for our **sum_s** signal, an overflow condition will result.

2. Save the changes to **Lab4.vhd** by clicking the **Save** button 💾 on the toolbar, then select the **Project** tab and right-click **Lab4.vhd** and select **Compile → Compile Selected.**

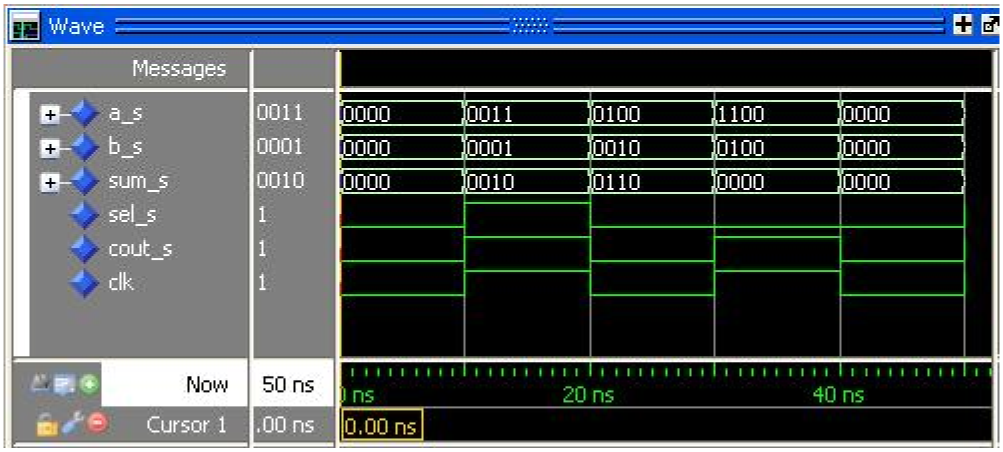


Project - H:/Altera_Training/Lab4/Lab4

Name	Status	Type	Order	Modified
And2.vhd	✓	VHDL	0	05/18/11 02:31:09 PM
fulladder.vhd	✓	VHDL	1	05/18/11 03:25:44 PM
Lab2.vhd	✓	VHDL	2	05/18/11 03:33:53 PM
Lab4.vhd	✓	VHDL	5	05/31/11 09:44:05 AM
Or3.vhd		VHDL	3	05/18/11 02:23:38 PM
Xor2.vhd		VHDL	4	05/18/11 02:23:37 PM

3. Once the compilation is complete, click the **Restart** button to restart the simulation, then click **Run** to run the simulation again.

4. Now it can be seen that the testing condition we added to the test bench file occurs from **30** to **40 ns** and the result is as expected, **cout_s** is high indicating overflow and **sum_s** is all low.



Lab 5

A VHDL Reaction Timer

Lab 5 – A VHDL Reaction Timer

This lab will combine many advanced VHDL techniques including timers, seven segment display drivers, packages and functions, and state machines to create a simple game that to test a person's reaction time.

Task 1: Create a New Project

1. **Start the New Project Wizard**
   Create a new project using the same procedure from the previous labs.
   Set the **Working Directory**, **Project Name**, and **Top-Level Design Entity** as seen in the table below:

Working Directory	H:\Altera_Training\Lab5
Project Name	Lab5
Top-Level Design Entity	Lab5

   *Note: After clicking **Next**, a window may pop up stating that the chosen working directory does not exist. Click **Yes** to create it.*
   Continue With the New Project Wizard using the same settings as before.

2. **Create a New File**
   Create a new VHDL file, as discussed previously.
   Copy and paste the following code into your new vhdl file and save it as **Lab5.vhd**:

Lab 5 – A VHDL Reaction Timer – Reaction Timer Code

```
library IEEE;
use IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.ALL;
USE work.bin_to_7seg.ALL;

ENTITY Lab5 IS
      PORT(
            -- 50Mhz clock signal from the DE2 board
            clk: IN std_logic;

            -- KEY3 and KEY0 on the DE2 board
            push_button, reset: IN std_logic;

            -- Outputs for the LEDS
            LEDR, LEDG: OUT std_logic_vector(7 DOWNTO 0);

            -- 7 segment display outputs
            digit1, digit2, digit3, digit4: OUT std_logic_vector(6 DOWNTO 0));

      END;

ARCHITECTURE Lab5_beh of Lab5 IS

      -- Type definition for the four states used for the state machine
      Type Timer_State IS (idle, countdown, timing, score);

      -- Signals of type Timer_State used to control the state machine
      SIGNAL current_state, next_state : Timer_State := idle;

      -- 16-bit binary value of the elapsed time in milliseconds
      SIGNAL elapsed_time: std_logic_vector(15 DOWNTO 0):= (OTHERS => '0');

      -- Flag that indicates the status of the countdown
      SIGNAL countdown_done: std_logic := '0';
      -- Flag that indicates a good input signal and the desire to change states
      SIGNAL state_change: std_logic := '0';

      -- Sample pulse generated by the SampleGen process used by the Debounce
      -- process to either increment the counter or signal a state change
      SIGNAL sample: std_logic;

      -- Maximum number of good input pulses needed to qualify as clean input
      CONSTANT PULSE_COUNT_MAX: integer := 20;

      -- Counter used to record the number of good input pulses
      SIGNAL counter: integer range 0 to PULSE_COUNT_MAX;

      -- The KEY0-3 push buttons on the DE2 board are active low, so sync is the
      -- inverse of the KEY3 push button's value
      SIGNAL sync: std_logic;

BEGIN

-------------------- SampleGen Process -------------------------
-- A counter that sends a signal to sample the input button
-- when the maximum count is reached.
SampleGen : process(clk)

      -- Sample_counter is a counter used to control the sample frequency
      -- sample frequency = clock frequency / (sample_counter(max)+1)
      variable sample_counter: integer range 0 to 24999;

begin
      if rising_edge(clk) then
            if (reset='0') then
                  sample      <= '0';
                  sample_counter := 0;
            else
                  if (sample_counter = 24999) then
                        sample_counter    := 0;
                        sample      <= '1';
                  else
                        sample      <= '0';
                        sample_counter := sample_counter + 1;

                  end if;
            end if;
      end if;
end process;

-------------------- Debounce Process -------------------------
-- A counter that is incrememnted every sample pulse while the
-- input button is pressed, when the switch is not pressed the
-- counter is reset to zero.  if the counter is at its maximum,
-- the debouncer output is high, otherwise its low.
Debounce : process(clk)
      begin
            if rising_edge(clk) then
                  if (reset='0') then
                        sync <= '0';
                        counter <= 0;
                        state_change <= '0';

                  else
                        sync <= NOT push_button;
                        if (sync='0') then            -- Button not pressed
                              counter <= 0;
                              state_change <= '0';
                        elsif(sample = '1') then      -- Button pressed
                              if (counter=PULSE_COUNT_MAX) then
                                    state_change <= '1';
                              else
                                    counter <= counter + 1;
                              end if;
                        end if;
                  end if;
            end if;
      end process;

-------------- Timer_State_Machine Process --------------------
-- Sets the next_state for the state machine.  The next
-- desired state is stored on the next_state signal, which is
-- used in the State_Transistions process to change the current
-- state.
Timer_State_Machine: PROCESS (clk)
      BEGIN
            next_state <= current_state;
            CASE current_state IS

      -- If the state_change flag is set and the push_button is pushed then the
      -- next desired state is the countdown state
                  WHEN idle =>
                        IF state_change = '1' AND push_button = '1' THEN
                              next_state <= countdown;
                        END IF;

      -- When the countdown_done flag is high the next desired state is the timing
      -- state
                  WHEN countdown =>
                        IF countdown_done = '1' THEN
                              next_state <= timing;
                        END IF;

      -- If the state_change flag is high and the push_button is pushed then the
      -- next desired state is the score state.
                  WHEN timing =>
                        IF state_change = '1' AND push_button = '1' THEN
                              next_state <= score;
                        END IF;

      -- Stay in score state.  Pressing reset will override this
                  WHEN score =>
                        next_state <= score;

      -- In case of entering any undefined states, next desired state is idle
                  WHEN OTHERS =>
                        next_state <= idle;

            END CASE;
      END PROCESS Timer_State_Machine;

--------------- State_Transitions Process ---------------------
-- Loads the next_state onto the current_state, or resets the
-- state to idle if the reset button is pressed.
State_Transitions: PROCESS (reset, clk)
      BEGIN
            IF reset = '0' THEN
                  current_state <= idle;
            ELSIF rising_edge (clk) THEN
                  current_state <= next_state;
            END IF;
      END PROCESS state_transitions;

----------------- Output_Process Process ----------------------
-- This process is what controls the LED the outputs as seen
-- by the user.
Output_Process: PROCESS (current_state, clk)

      -- Two 16-bit countdown counters to create a delay for the countdown state
      variable countdown_count1, countdown_count2: integer  range 0 to 65535;

      -- A delay counter used to delay the incrementing of timing_count so that
      -- timing_count is in milliseconds
      variable timing_counter: integer range 0 to 49999;

      -- 16-bit binary value of the time count in milliseconds
      variable timing_count: std_logic_vector(15 DOWNTO 0);
      BEGIN
      IF rising_edge(clk) THEN
            CASE current_state IS

 -- Reset everything in idle mode
                  WHEN idle =>
                        LEDG <= (OTHERS => '0');
                        LEDR <= (OTHERS => '0');
                        elapsed_time <= (OTHERS => '0');
                        timing_count := (OTHERS => '0');
                        timing_counter := 0;
                        countdown_count1 := 0;
                        countdown_count2 := 0;

 -- Turn red LEDs on and begin countdown counter, when countdown counter
 -- is done, countdown_done is high.
                  WHEN countdown =>
                        LEDG <= (OTHERS => '0');
                        LEDR <= (OTHERS => '1');

                        countdown_count1 := countdown_count1 + 1;
                        IF countdown_count1 = 65535 THEN
                              countdown_count2 := countdown_count2 + 1;
                              countdown_count1 := 0;
                        END IF;
                        IF (countdown_count2 = 763) THEN
                              countdown_done <= '1';
                              countdown_count2 := 0;
                        END IF;

 -- Turn green LEDs on and begin counting in milliseconds.  Set countdown_done
 -- to low.
                  WHEN timing =>
                        LEDG <= (OTHERS => '1');
                        LEDR <= (OTHERS => '0');

                        IF (timing_counter = 49999) THEN
                              timing_count := timing_count + 1;
                              timing_counter := 0;
                        ELSE
                              timing_counter := timing_counter + 1;
                        END IF;
                        elapsed_time <= timing_count;
                        countdown_done <= '0';

 -- Turn all LEDs on, set timing counters to 0
                  WHEN score =>
                        LEDG <= (OTHERS => '1');
                        LEDR <= (OTHERS => '1');

                        timing_count := (OTHERS => '0');
                        timing_counter := 0;

 -- Turn all LEDs off
                  WHEN OTHERS =>
                        LEDG <= (OTHERS => '0');
                        LEDR <= (OTHERS => '0');
            END CASE;
      END IF;
      END PROCESS Output_Process;

------------------- Time_Display Process -----------------------
-- This process displays the time value on four of the seven
-- segment displays on the DE2 board.  The functions bin_to_bcd
-- and bcd_to_7seg are called from the included bin_to_7seg.vhd
Time_Display_Process: PROCESS (elapsed_time)

      -- 20-bit binary variable that holds 5 4-bit BCD values, only 4 are used to
      -- display on the 4 7-segment displays
      variable bcd_time: std_logic_vector(19 DOWNTO 0);

      BEGIN
      -- First then elapsed time must be converted from binary to BCD using the
      -- bin_to_bcd() function and stored on to the bcd_time variable.
      -- Next, the BCD values are fed out to the four 7-segment displays by using
      -- the bcd_to_7seg() function.  The first four bits of bcd_time correspond
      -- to digit1, the next four to digit2, and so on.

            bcd_time := bin_to_bcd(elapsed_time);
            digit1 <= bcd_to_7seg(bcd_time(3 DOWNTO 0));
            digit2 <= bcd_to_7seg(bcd_time(7 DOWNTO 4));
            digit3 <= bcd_to_7seg(bcd_time(11 DOWNTO 8));
            digit4 <= bcd_to_7seg(bcd_time(15 DOWNTO 12));


      END PROCESS;

END Lab5_beh;
```

3. **Create another New File**
   Create another new VHDL file.
   Copy and paste the following code into your new vhdl file and save it in the project directory as **bin_to_7seg.vhd**, make sure the **Add file to current project** box is checked in the **Save As** window:

Lab 5 – A VHDL Reaction Timer – Reaction Timer Code

```
library IEEE;
use IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.ALL;
use ieee.numeric_std.all;

PACKAGE bin_to_7seg IS

      -- Bin_to_BCD takes in a 16 bit binary number and returns a 20-bit BCD
      -- representation of it.  This would be the equivalent of a 5-digit integer
      -- with 5 groups of 4-bits, each representing 1 digit.
      FUNCTION Bin_to_BCD (bin :std_logic_vector(15 DOWNTO 0)) return
std_logic_vector;

      -- BCD_to_7seg takes a 4-bit BCD value and returns a 7-bit value that
      -- represents the desired segments to turn on for a 7-segment display
      FUNCTION BCD_to_7seg (bcd :std_logic_vector(3 DOWNTO 0)) return
std_logic_vector;

END;

PACKAGE BODY bin_to_7seg IS

------------------- Binary to BCD Conversion Function ------------------
      FUNCTION Bin_to_BCD (bin :std_logic_vector(15 DOWNTO 0)) return
std_logic_vector IS
            variable i : integer:=0;
            variable bcd : std_logic_vector(19 downto 0) := (others => '0');
            variable bint : std_logic_vector(15 DOWNTO 0) := bin;

                  BEGIN

                        for i in 0 to 15 loop
                              bcd(19 downto 1) := bcd(18 downto 0);
                              bcd(0) := bint(15);
                              bint(15 downto 1) := bint(14 downto 0);
                              bint(0) :='0';

                              if(i < 15 and bcd(3 downto 0) > "0100") then
                                    bcd(3 downto 0) := bcd(3 downto 0) + "0011";
                              end if;

                              if(i < 15 and bcd(7 downto 4) > "0100") then
                                    bcd(7 downto 4) := bcd(7 downto 4) + "0011";
                              end if;
                              if(i < 15 and bcd(11 downto 8) > "0100") then
                                    bcd(11 downto 8) := bcd(11 downto 8) + "0011";
                              end if;

                              if(i < 15 and bcd(15 downto 12) > "0100") then
                                    bcd(15 downto 12) := bcd(15 downto 12) + "0011";
                              end if;

                              if(i < 15 and bcd(19 downto 16) > "0100") then
                                    bcd(19 downto 16) := bcd(19 downto 16) + "0011";
                              end if;

                        END LOOP;

                        RETURN BCD;
                  END FUNCTION;

------------------      BCD to 7 Seg Display Conversion Function ------------------
      FUNCTION BCD_to_7Seg (bcd :std_logic_vector(3 DOWNTO 0)) return
std_logic_vector IS

                  variable seg7 : std_logic_vector(6 downto 0) := (others => '1');

                  BEGIN
                        CASE bcd IS
                              WHEN "0000"=>seg7:="1000000"; -- 0 --
                              WHEN "0001"=>seg7:="1111001"; -- 1 --
                              WHEN "0010"=>seg7:="0100100"; -- 2 --
                              WHEN "0011"=>seg7:="0110000"; -- 3 --
                              WHEN "0100"=>seg7:="0011001"; -- 4 --
                              WHEN "0101"=>seg7:="0010010"; -- 5 --
                              WHEN "0110"=>seg7:="0000010"; -- 6 --
                              WHEN "0111"=>seg7:="1111000"; -- 7 --
                              WHEN "1000"=>seg7:="0000000"; -- 8 --
                              WHEN "1001"=>seg7:="0011000"; -- 9 --
                              WHEN OTHERS=>seg7:="1111111"; -- X --
                        END CASE;
                        RETURN seg7;
                  END FUNCTION;

END PACKAGE BODY;
```

4. **About the Code**
   The code in **Lab5.vhd** is broken up into a variety of processes.
   SampleGen and **Debounce** are processes used together to debounce the input signal from the momentary push button (**KEY3**) on the development board. The hardware samples the push button at a rate of 2 kHz, and after 20 consecutive low readings (in this case the push button switches are active low) the hardware considers the button to have been pressed.

   The program uses a finite state machine to represent the various states used. These states are **idle**, **countdown**, **timing**, and **score**. **Idle** is when the hardware is waiting for the user to initiate the game. **Countdown** is a timed countdown sequence which adds a delay before the millisecond timer starts counting. **Timing** is when the millisecond timer starts counting how long it takes the user to push the button to stop the counting. **Score** freezes the display to show the user's reaction time.

   The process **Timer_State_Machine** is what defines the requirements to change from one state to the next. This process stores the next desired state onto the **next_state** signal, which is then set as the **current_state** by the **State_Transitions** process.

   The **Output_Process** contains everything that happens within each of the individual states.
   In the **Idle** state within the the green and red LEDs are turned off, and all timers and counters are reset to zero.
   In the **Countdown** state the red LEDs are turned on and the countdown timer starts. When the

countdown timer is done, **countdown_done** is set high.
Once **countdown_done** is high, the **current_state** becomes the **Timing** state. Here the millisecond timer starts counting until the user presses the button to stop it.
Finally in the **Score** state, the user's time is displayed in milliseconds.

Pressing the reset button (**KEY1** on the DE2 development board) will return you to the **idle** state at any point.

The **Time_Display_Process** takes in the elapsed time in binary format, converts that from binary to BCD, and then to a value suitable to use to display the desired decimal digit on one of four seven segment displays.

The **bin_to_7seg.vhd** file contains two functions used by the **Lab5.vhd** code within the **Time_Display_Process**.
The first function is **Bin_to_BCD**, which uses a double dabble algorithm to convert a 16-bit binary value to five 4-bit BCD values, although only four of these BCD values are used by the **Lab5.vhd** code.
The other function, **BCD_to_7seg**, converts the BCD values to a 7-bit value that represents each digit for the seven segment displays.

As part of this lab exercise, in the **Lab5.vhd** file, you will need to finish the case statement in the **Timer_State_Machine** process, as well as the function calls in the **Time_Display_Process** used to display the time. Some hints are provided in the comments in the code.

Task 2: Pin Assignments

1. **Make the Following Pin Assignments**

	To	Location	DE2 Board Description
1	push_button	PIN_W26	KEY3
2	reset	PIN_N23	KEY1
3	digit1[0]	PIN_AF10	Seven Segment Digit 0[0]
4	digit1[1]	PIN_AB12	Seven Segment Digit 0[1]
5	digit1[2]	PIN_AC12	Seven Segment Digit 0[2]
6	digit1[3]	PIN_AD11	Seven Segment Digit 0[3]
7	digit1[4]	PIN_AE11	Seven Segment Digit 0[4]
8	digit1[5]	PIN_V14	Seven Segment Digit 0[5]
9	digit1[6]	PIN_V13	Seven Segment Digit 0[6]
10	digit2[0]	PIN_V20	Seven Segment Digit 1[0]
11	digit2[1]	PIN_V21	Seven Segment Digit 1[1]
12	digit2[2]	PIN_W21	Seven Segment Digit 1[2]
13	digit2[3]	PIN_Y22	Seven Segment Digit 1[3]
14	digit2[4]	PIN_AA24	Seven Segment Digit 1[4]
15	digit2[5]	PIN_AA23	Seven Segment Digit 1[5]
16	digit2[6]	PIN_AB24	Seven Segment Digit 1[6]
17	digit3[0]	PIN_AB23	Seven Segment Digit 2[0]
18	digit3[1]	PIN_V22	Seven Segment Digit 2[1]
19	digit3[2]	PIN_AC25	Seven Segment Digit 2[2]
20	digit3[3]	PIN_AC26	Seven Segment Digit 2[3]
21	digit3[4]	PIN_AB26	Seven Segment Digit 2[4]
22	digit3[5]	PIN_AB25	Seven Segment Digit 2[5]

23	digit3[6]	PIN_Y24	Seven Segment Digit 2[6]
24	digit4[0]	PIN_Y23	Seven Segment Digit 3[0]
25	digit4[1]	PIN_AA25	Seven Segment Digit 3[1]
26	digit4[2]	PIN_AA26	Seven Segment Digit 3[2]
27	digit4[3]	PIN_Y26	Seven Segment Digit 3[3]
28	digit4[4]	PIN_Y25	Seven Segment Digit 3[4]
29	digit4[5]	PIN_U22	Seven Segment Digit 3[5]
30	digit4[6]	PIN_W24	Seven Segment Digit 3[6]
31	clk	PIN_N2	50 MHz on board clock
32	LEDG[0]	PIN_AE22	LED Green [0]
33	LEDG[1]	PIN_AF22	LED Green [1]
34	LEDG[2]	PIN_W19	LED Green [2]
35	LEDG[3]	PIN_V18	LED Green [3]
36	LEDG[4]	PIN_U18	LED Green [4]
37	LEDG[5]	PIN_U17	LED Green [5]
38	LEDG[6]	PIN_AA20	LED Green [6]
39	LEDG[7]	PIN_Y18	LED Green [7]
40	LEDR[0]	PIN_AE23	LED Red [0]
41	LEDR[1]	PIN_AF23	LED Red [1]
42	LEDR[2]	PIN_AB21	LED Red [2]
43	LEDR[3]	PIN_AC22	LED Red [3]
44	LEDR[4]	PIN_AD22	LED Red [4]
45	LEDR[5]	PIN_AD23	LED Red [5]
46	LEDR[6]	PIN_AD21	LED Red [6]
47	LEDR[7]	PIN_AC21	LED Red [7]

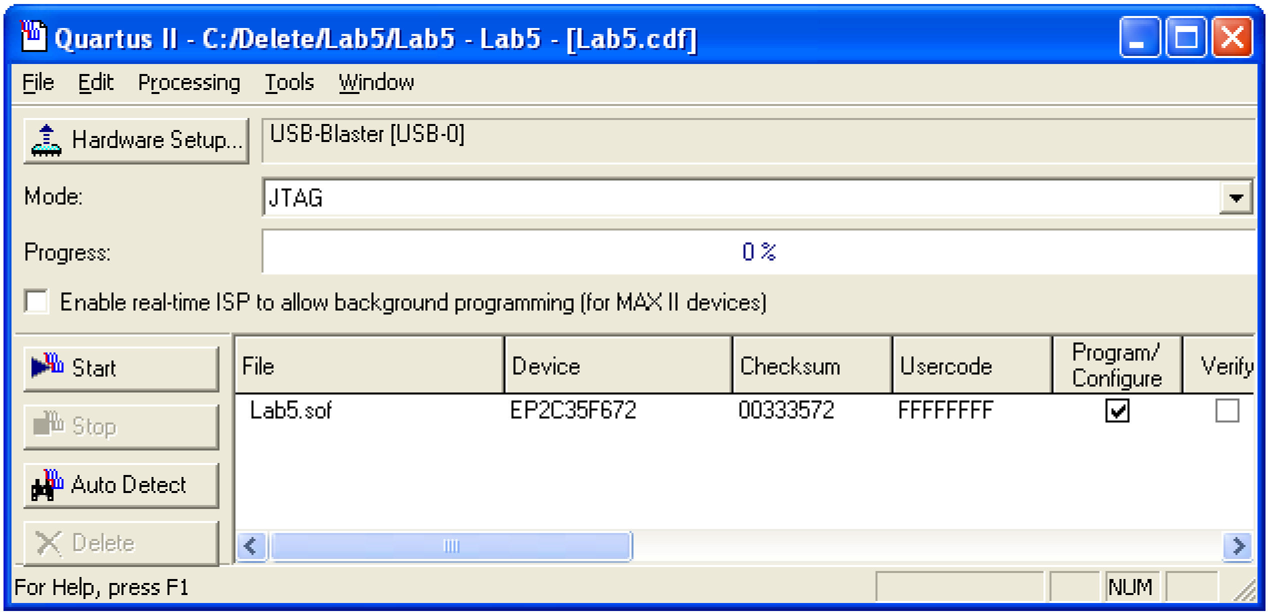
2. Click **Save** 💾 to save the pin assignments.

3. **Compile the Design** by clicking on the **Start Compilation** button ▶

Task 3: Program the DE2 Development Board

1. Plug in and power on the DE2 board. Make sure that the **RUN/PROG Switch for JTAG/AS Modes** is in the **RUN** position.

2. In the Quartus II window click the **Programmer** button on the Toolbar to open the Programmer window
The **Hardware Setup…** must be **USB-Blaster [USB-0]**. If not, click the **Hardware Setup…** button and select **USB-Blaster [USB-0]** from the drop-down menu for **Currently selected hardware**.
Mode should be set to **JTAG**.
Make sure that the **File** is **Lab5.sof**, **Device** is **EP2C35F672**, and the **Program/Configure** box is checked.



Quartus II - C:/Delete/Lab5/Lab5 - Lab5 - [Lab5.cdf]

Hardware Setup... USB-Blaster [USB-0]

Mode: JTAG

Progress: 0 %

☐ Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify
Lab5.sof	EP2C35F672	00333572	FFFFFFFF	☑	☐

For Help, press F1

Then click the **Start** button to program the DE2 board.
When the progress bar reaches 100%, programming is complete

Task 4: Play the Game

1. The game is started by pressing **KEY3** on the development board. This causes the hardware to change states from **idle** to **countdown**, turns on the first eight red LEDs, and begins the countdown to when the timer starts. The countdown is not displayed in order to maintain the element of surprise.

2. When the countdown completes the hardware changes to the **timer** state and begins incrementing the millisecond counter, the red LEDs are shut off, and the first eight green LEDs are turned on. The user must press **KEY3** on the development board as soon as possible after the green LEDs light in order to obtain the quickest reaction time.

3. When the user presses **KEY3** the state is changed to the **score** state where the millisecond display is frozen in order to display the user's reaction time.

4. Pressing the **reset** button (**KEY1**) at this point will return the hardware to the **idle** state where the game can be played again.

National Science Foundation – Advanced Technological Education
VHDL and FPGA design workshop – (Pre - Course Evaluation)
Summer 2011

Please take a few minutes to answer the following questions. Your responses will be used to improve this course for future students. Thank you, in advance, for your participation.

Mastery of Main Course Objectives (Circle One)

	Complete Mastery				No Mastery
Ability to implement basic constructs of VHDL	5	4	3	2	1
Ability to implement modeling structures of VHDL	5	4	3	2	1
Ability to use software tools to check the code for correctness and correct errors	5	4	3	2	1

Additional content/topics you would have liked to have had covered ________________

____

____

Quality of Instruction (Circle One)

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
The instruction was clearly presented	5	4	3	2	1
Any questions I asked were properly answered	5	4	3	2	1
The materials provided helped me to learn	5	4	3	2	1
The pace of the course was appropriate for the amount of material to be learned	5	4	3	2	1
All things taken into consideration, I considered the instruction to be excellent					

Additional comments regarding the teaching of this course or suggestions for improvement

____

____

Advanced VHDL Design Technique Learning Objectives (Circle One)

You learned to:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Write synthesizable VHDL	5	4	3	2	1
Control state machine implementation	5	4	3	2	1
Optimize a design using operator balancing, resource sharing, and pipelining	5	4	3	2	1
Create a test bench and run a simulation	5	4	3	2	1

Additional comments regarding the teaching of this material

Quartus II Software Design Series: Foundation Learning Objectives (Circle One)

You learned to:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Create a new Quartus II project	5	4	3	2	1
Create design components using MegaWizard manager	5	4	3	2	1
Compile a design and view results	5	4	3	2	1
Use settings and assignments to control results	5	4	3	2	1
Make pin assignments and evaluate	5	4	3	2	1
Use the TimeQuest timing analyzer	5	4	3	2	1

Additional comments regarding the teaching of this material

National Science Foundation – Advanced Technological Education
VHDL and FPGA design workshop – (Post - Course Evaluation)
Summer 2011

Please take a few minutes to answer the following questions. Your responses will be used to improve this course for future students. Thank you, in advance, for your participation.

Mastery of Main Course Objectives (Circle One)

	Complete Mastery				No Mastery
Ability to implement basic constructs of VHDL	5	4	3	2	1
Ability to implement modeling structures of VHDL	5	4	3	2	1
Ability to use software tools to check the code for correctness and correct errors	5	4	3	2	1

Additional content/topics you would have liked to have had covered ________________
____
____

Quality of Instruction (Circle One)

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
The instruction was clearly presented	5	4	3	2	1
Any questions I asked were properly answered	5	4	3	2	1
The materials provided helped me to learn	5	4	3	2	1
The pace of the course was appropriate for the amount of material to be learned	5	4	3	2	1
All things taken into consideration, I considered the instruction to be excellent					

Additional comments regarding the teaching of this course or suggestions for improvement
____
____

Advanced VHDL Design Technique Learning Objectives (Circle One)

You learned to:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Write synthesizable VHDL	5	4	3	2	1
Control state machine implementation	5	4	3	2	1
Optimize a design using operator balancing, resource sharing, and pipelining	5	4	3	2	1
Create a test bench and run a simulation	5	4	3	2	1

Additional comments regarding the teaching of this material

____

____

Quartus II Software Design Series: Foundation Learning Objectives (Circle One)

You learned to:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Create a new Quartus II project	5	4	3	2	1
Create design components using MegaWizard manager	5	4	3	2	1
Compile a design and view results	5	4	3	2	1
Use settings and assignments to control results	5	4	3	2	1
Make pin assignments and evaluate	5	4	3	2	1
Use the TimeQuest timing analyzer	5	4	3	2	1

Additional comments regarding the teaching of this material

____

____